PROJECTS THEORY APPLICATIONS CIRCUITS TECHNOLOGY

NUTS AND VOLTS

www.nutsvolts.com
June 2016

EVERYTHING FOR ELECTRONICS

ANALOG DISPLAY For the INTERNET of THINGS

MAKE A SIMPLE network time protocol clock

THE BASICS of RF signal propagation

UNRAVELING the mysteries of ground


SERVOCITY.COM
JOIN OUR COMMUNITY
WE HAVE THE PARTS FOR YOUR IDEAS
MODIFY
USE THESE HANDY TOOLS TO CUSTOMIZE SERVOS FOR SPECIFIC APPLICATIONS
Servo Stretcher $19.99
HFP-25 Programmer $69.99
NEW DPC-11 Programmer $19.99
Servo Protractor $17.99
Servo Mixer $9.95
Servo Reverser $39.99
Servo Power Boards $9.99 - $27.99
ACCESSORIZE
UNIQUE AND EXCLUSIVE SERVO ACCESSORIES
Parallel Gripper Kit $14.99
$9.99 Standard Gripper Kit A
$3.08 - $14.99 Servo Gears
Servo Sprockets $3.85 - $12.99
$3.99 Servo Hubs
Servo Arms
PATENTED
IMPROVE
MORE POWER
MORE STRENGTH
MORE RESOLUTION
ServoBlocks™ $24.99 - $28.99
PATENTED
INCREASE THE LOAD-BEARING CAPABILITIES OF YOUR SERVOS
[Fits 24T & 25T Standard Size Servos]
Boosted Extensions $13.99 - $15.99
Servo Gearboxes starting at $49.99
[Top Mount]
[Tube Mount]
[Bottom Mount]
[Channel Mount]
CONTROL
PRECISION CONTROL + HELPFUL FEATURES AT AN AFFORDABLE PRICE
2 Servo Joystick $99.99
4 Servo Recorder $109.99
[Kit] [Assembled]
NEW Dual Servo Driver $29.99 or $49.99
[Assembled] [Kit]
NEW Dual Servo Controller $59.99 or $79.99
LET US DO THE WORK FOR YOU - WE MODIFY SERVOS!
sales@servocity.com | 620.221.0123 | www.servocity.com
Copyright 1999 - 2016 Robotzone, LLC - All rights reserved. ServoCity® and Actobotics® are registered trademarks of Robotzone, LLC

June 2016

Subscription Information
Nuts & Volts — PO Box 15277
North Hollywood, CA 91615-9218
Call 877-525-2539 or go to www.nutsvolts.com
Subscribe • Gift • Renewal • Change of Info

Page 12

Page 24

Departments

Nuts & Volts (ISSN 1528-9885/CDN Pub Agree #40702530) is published monthly for $26.95 per year by T & L Publications, Inc., 430 Princeland Court, Corona, CA 92879. PERIODICALS POSTAGE PAID AT CORONA, CA AND AT ADDITIONAL MAILING OFFICES. POSTMASTER: Send address changes to **Nuts & Volts, P.O. Box 15277, North Hollywood, CA 91615** or Station A, P.O. Box 54, Windsor ON N9A 6J5; cpcreturns@nutsvolts.com.

DEVELOPING PERSPECTIVES

by Bryan Bergeron, Editor

Wall Light Switches: Relics of the Tungsten Age?

Given the increased popularity of multi-function light bulbs, it's clear that the traditional light-only bulb and the associated 110V circuitry are on their way out. I'm not talking about the compact fluorescent (CFC) or even LED "replacement" bulbs, but smart bulbs that do much more than produce heat and light.

I replaced the Tungsten bulbs in my home with 500K or daylight CFC bulbs almost a decade ago. It was an expensive upgrade; in part because the original Tungsten bulbs were still perfectly functional. About a year ago, I started replacing the CFC bulbs with LED bulbs. Again, I tossed completely functional fluorescent bulbs to move up to a cooler operating/more compact light bulb. An added feature was the ability to dim the LED bulbs — something I couldn't do with a standard CFC.

More recently, I upgraded several of the CFC light bulbs to multi-color LED bulbs that I can operate from my Apple iOS device. With a simple app, I can change the brightness and hue of the lights, set a timer to wake me with light, and operate the lights when I'm away from home. The technology has been around for years, but I'm just getting to the point where I no longer need to look for the light switch when entering a room. My Wi-Fi enabled light bulbs are always on, awaiting my next command. As such, there isn't a need for the light switch.

My latest journey in light bulb technology does more than simply replace one light source for another. No, the latest generation of always-on "light bulb" replacements makes use of the house wiring and light fixtures, and happens to produce light almost as an afterthought.

For example, Sengled (available at Home Depot and Amazon) offers an integrated microphone/speaker LED bulb that plugs into a standard socket. With the proper peripherals, the bulb supports voice control of cloud connected devices, as well as the ability to detect glass breaking. The Sengled Pulse base serves as a Bluetooth speaker ($150/pair) that is by no means cheap when compared with a standard battery-powered Bluetooth speaker. I found the Pulse to be the ultimate in a low clutter stereo speaker setup.

Then, there are the Wi-Fi repeater bulbs which replace the clunky plug-in desktop repeaters.

At the top on my wish-list for future light bulb "replacements" is an odor detector bulb for my refrigerator that emails me when produce or milk products go bad. I also want an emergency flashlight with a bulb that automatically dials 911 at the press of a button. There are replacement car headlights and tail lights that provide collision avoidance detection, as well. I can even envision a doctor's penlight that doubles as an optical test device that can diagnose a variety of eye conditions.

As manufacturers are proving, just about any electronic device imaginable can be made to fit the size and power limitations of a traditional screw-in light bulb. I expect the typical technology leapfrogging, with superior offerings from the likes of Philips, GE, and eventually Apple.

Anyone interested in a slightly used set of first generation smart bulbs? **NV**

NUTS AND VOLTS

Published Monthly By
T & L Publications, Inc.
430 Princeland Ct.
Corona, CA 92879-1300
(951) 371-8497
FAX **(951) 371-3052**
Webstore orders only **1-800-783-4624**
www.nutsvolts.com

Subscription Orders
Toll Free **1-877-525-2539**
Outside US **1-818-487-4545**
P.O. Box 15277
North Hollywood, CA 91615

FOUNDER
Jack Lemieux

PUBLISHER
Larry Lemieux
publisher@nutsvolts.com

ASSOCIATE PUBLISHER/ ADVERTISING SALES
Robin Lemieux
robin@nutsvolts.com

EDITOR
Bryan Bergeron
techedit-nutsvolts@yahoo.com

VP OF OPERATIONS
Vern Graner
vern@nutsvolts.com

CONTRIBUTING EDITORS
Fred Eady, Tim Brown, Lou Frenzel, Paul Verhage, Chuck Hellebuyck, David Goodsell, John Weit, John Rucker, Jeff Tregre, Craig Lindley

CIRCULATION DEPARTMENT
subscribe@nutsvolts.com

SHOW COORDINATOR
Audrey Lemieux

WEBSTORE MARKETING COVER GRAPHICS
Brian Kirkpatrick
sales@nutsvolts.com

WEBSTORE MANAGER/ PRODUCTION
Sean Lemieux
sean@nutsvolts.com

ADMINISTRATIVE STAFF
Re Gandara

Reader Feedback

Our Bad!

In the May 2016 Design Cycle column, Lemos International's website was incorrectly listed. It should be **www.lemosint.com**. We apologize for any inconvenience this caused!

Vintage Versus New

I read Bryan Bergeron's recent editorial on vintage repair of a Singer sewing machine with a broad grin on my face. This machine demonstrated a brilliant but simple design constructed with quality materials.

I am familiar with the Singer sewing machine since our repair shop began in 1965, and we serviced kitchen and household appliances. I remember the Sunbeam toaster where two pieces of bread are placed in position and slowly disappear, then two perfectly browned pieces of toast automatically pop up.

A note to readers who are not familiar with the Singer and Sunbeam products — they do not use any semiconductor, microprocessor, PIC, or special coded instructions to be functional.

The Singer and Sunbeam products were not cheap. They were expected to last 10-15 years or more. If any product required repair, they were expected to last another 10-15 years.

To survive, our TV shop had to change as we now service electronic items which are primarily TVs. At any time, we normally have at least 50 or more TVs in for repair and most are under 5-7 years old; some are under two years old. I wonder what is missing from the design criteria which results in reduced reliability.

Back to the sewing machine and toaster. Today's toaster lasts less than two years and makes crummy toast. Sewing machines today are microprocessor controlled and do many different stitch variations. Let's see how long they last.

Dave and Ken Walker
Dependable Electronic TV Service
Tustin, CA

Tube Amps and Analog Computers

In the May 2016 Reader Feedback from James Kretzschmar, I noticed that he is using 1625s in his audio amplifier. The stereo amplifier I built in the early '60s used 1625s (for the uninitiated, the 1625 is a power tetrode similiar to an 807 but with a seven-pin base and 12 volts on the filament) that I purchased from Burstein-Applebee for 25

Continued on page 63

In this column, Tim answers questions about all aspects of electronics, including computer hardware, software, circuits, electronic theory, troubleshooting, and anything else of interest to the hobbyist. Feel free to participate with your questions, comments, or suggestions. **Send all questions and comments to: Q&A@nutsvolts.com.**

- **Microwave Oven or Death Ray?**
- **Modem Commands**
- **Mailbag**

Microwave Oven or Death Ray?

Q My microwave oven has a glass window on the door so that I can see my food inside, which is nice. So, how do they keep the dangerous microwave radiation from coming through the glass?

Justin Langston
Killeen, TX

A Actually, manufacturers embed nano-gremlins in the glass that use nano-baseball bats to knock the microwaves back into the oven – NOT! Here's a little history, science, and the answer to your question.

The technology which gives us the microwave oven was accidentally discovered in 1945 by Percy Spencer who was working on a radar unit at Raytheon and noticed that the microwaves from the antenna melted a candy bar in his pocket. There is an old rumor that Raytheon engineers would heat their sandwiches by placing them in front of a radar antenna.

Microwave ovens were only used commercially until the late 1970s when they became available for residential use. I remember the units in a plant cafeteria in 1968. We finally brought a huge sized unit home in the early 1980s for a cost around $1,500 (now you can find much better units for $50 to $100 depending on the bells and whistles you want). In the mid '80s, I repaired an oven that was blowing a fuse by replacing a capacitor for a cost of $10, which was much less than the cost of a new oven at the time.

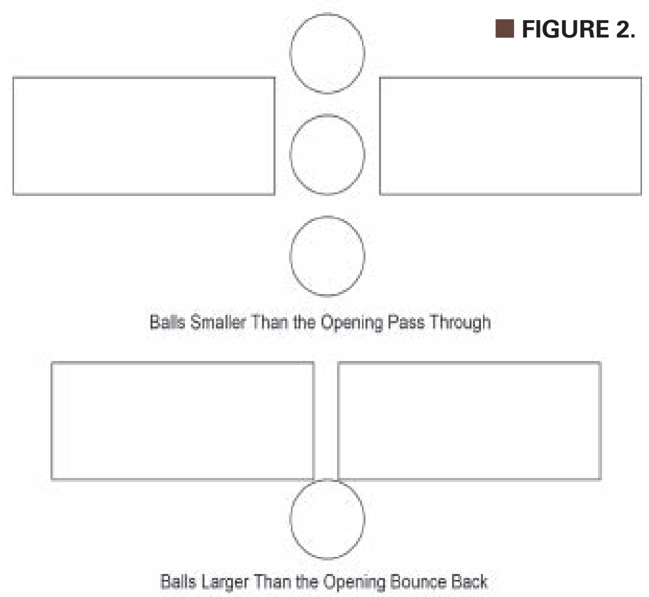
The microwave oven heats foods by flooding the inside of the unit with electromagnetic waves at a frequency of 2.45 GHz (wavelength of 12.2 centimeters) and a power of 700 watts for a 1,100 watt unit. These rapidly changing waves alternate the electric field polarity between positive and negative, thus causing the water molecules to rotate between the positive hydrogen ends and negative oxygen ends trying to align with the electric field. This motion of the water molecules heats the food where the water molecules are located. Ever wonder why sometimes the plate gets hot while the food is still cold? The plate has a lot of absorbed water molecules that heat up and the plate conducts the heat better than the food. This is why you should always use dishes marked "Microwave Safe."

Now for the answer to how the microwave radiation stays in the oven and does not penetrate the glass. First, look at a typical electromagnetic wave as shown in **Figure 1**. The E-M wave has the electric and magnetic fields that oscillate in planes that are at 90 degrees to each other. Each E-M wave has an associated wavelength which depends on the frequency of the wave by the equation:

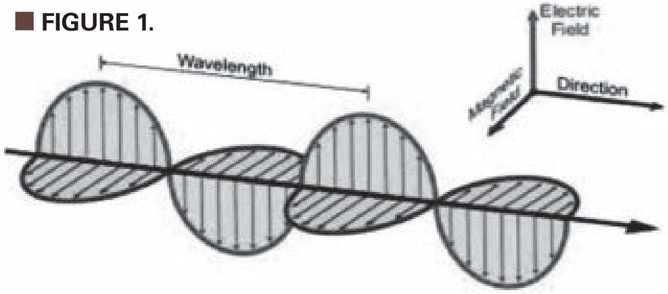
$$\lambda = c/f$$

where λ is the wavelength in centimeters, c is the speed of light (30 billion centimeters per second), and f is frequency in Hertz.

■ FIGURE 2.



■ FIGURE 1.



Post comments on this article at
www.nutsvolts.com/magazine/article/June2016-QA

Wavelength – and not amplitude – of the E-M wave is what is important when you look at the propagation of E-M waves. E-M waves will be reflected from any surface that has openings that are much smaller than the wavelength of the E-M wave. **Figure 2** shows an example of this reflection using balls in the place of the E-M waves. Balls (and E-M waves) will pass through an opening that is larger than the diameter of the ball (E-M wave wavelength), and the ball will bounce (E-M wave will reflect) from an opening that is smaller than the diameter of the ball (wavelength of the E-M wave).

The microwave door has holes with diameters of approximately one millimeter (which is 0.1 centimeters). The microwave E-M wavelength is about 120 times the size of these holes, so the microwave E-M wave bounces back into the oven instead of passing through the door. However, light waves with much smaller wavelengths (40 to 70 millionths of a centimeter) pass through the door.

The cabinet of the microwave oven is metal which reflects the microwaves around the inside of the oven, and the door gaskets are made of a conductive material which reflects the E-M waves back into the oven cavity. The inside of a microwave is an inverted Faraday cage which amateur radio operators use to keep electromagnetic interference out of their radios.

Under the plastic cover inside a microwave oven door you will see a series of cavities made by bending in tabs. These cavities are an odd multiple of one-quarter of a wavelength deep and are closed at the bottom of each cavity. These cavities act as an open circuit at the open end (inside of cover) which prevents microwave radiation from passing through them. **Figure 3** shows a series of transmission line/cavity configurations, in which (c) is the shorted quarter wave cavity which has a very large (theoretically infinite) input resistance.

Tim Brown

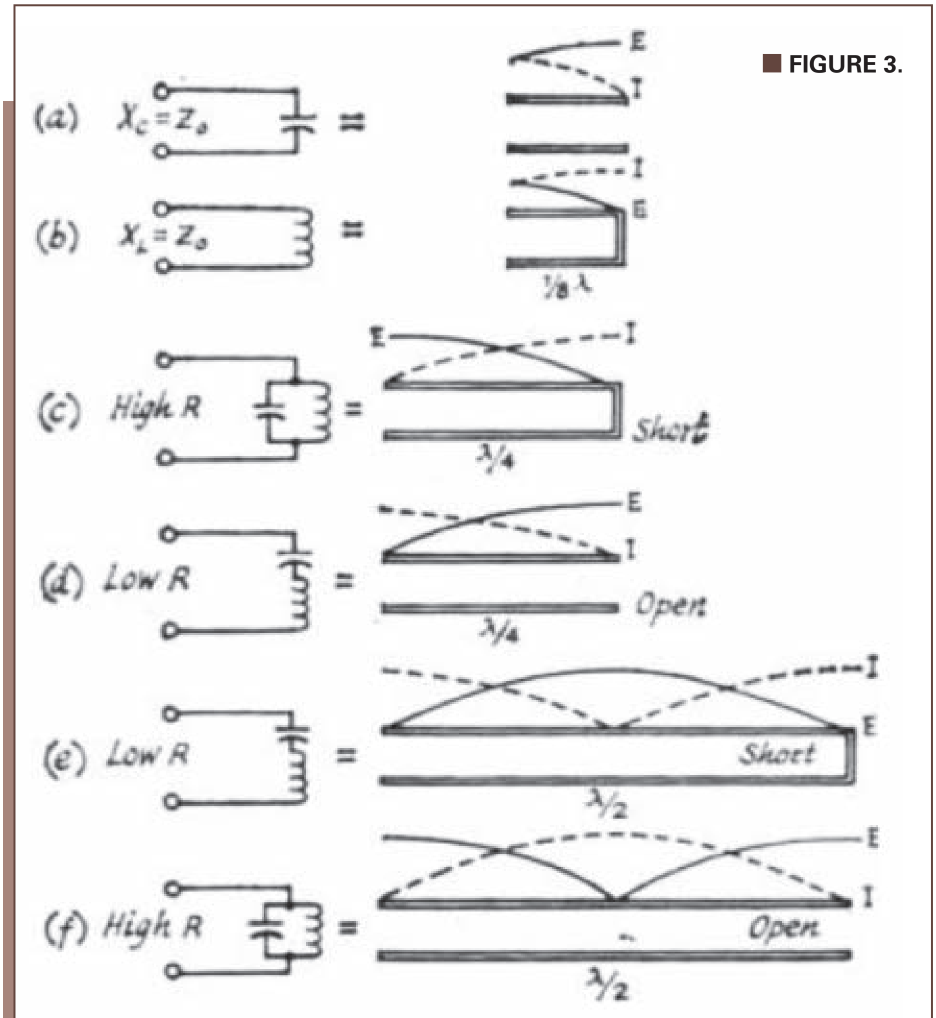


FIGURE 3.

Modem Commands

Q On modern modems and systems where caller ID is enabled, what would be the AT commands to output the incoming phone number as text to a file, to a display, etc.? I have been researching this for quite some time and can find no answer.

Danny Lusk
Bud, WV

A AT Commands or Attention Commands are used to control the operation of a modem from a PC. There are a variety of AT commands such as Caller ID Enable/Disable, Distinctive Ring Report, On Hold Initiate/Timer, and Callback Control, just to name a few. The AT codes are variations of those used on the Hayes SmartModem which was first marketed back in 1981.

The AT command for Caller ID is some variation of CID = 0 to disable Caller ID, and CID = 1 to enable Caller ID. The exact syntax of the AT commands varies with different modems, so consult your modem's manual for the proper syntax. (See Q&A SIDELINES for several examples

of modem manuals showing the AT commands.)

In order to read the caller ID on your PC, you will need software which takes the incoming Caller ID from the modem and displays it on the computer. Most of the Caller ID software allows you to screen and/or block telemarketing calls, which is great since the Do Not Call List does not work (at least not for us).

Some of the Caller ID software packages will let you "repair" inconsistencies in the Caller ID display caused by the way the modem presents data. The Caller ID software stores the call information, so you can export this information to another software platform for further processing.

See Q&A SIDELINES for information on AT commands and Caller ID software.

Tim Brown

MAILBAG

Re: Furnace Data Acquisition

I read with interest the comments regarding Furnace Data Acquisition in the February 2016 issue. For the last couple of years, I have been monitoring my own hydronic heating system using an opto-isolator across the 24 VAC thermostat contacts. The opto-isolator output, as well as two DS18B20 temperature sensors (one by the thermostat and one outside), are read by a PICAXE 08M2 which then sends the data every minute via serial output to a logging program running on a Windows XP PC. I've included a schematic of the thermostat circuitry which specifies a 4N25 opto-isolator; my setup uses a house marked IC, but the 4N25 should be similar.

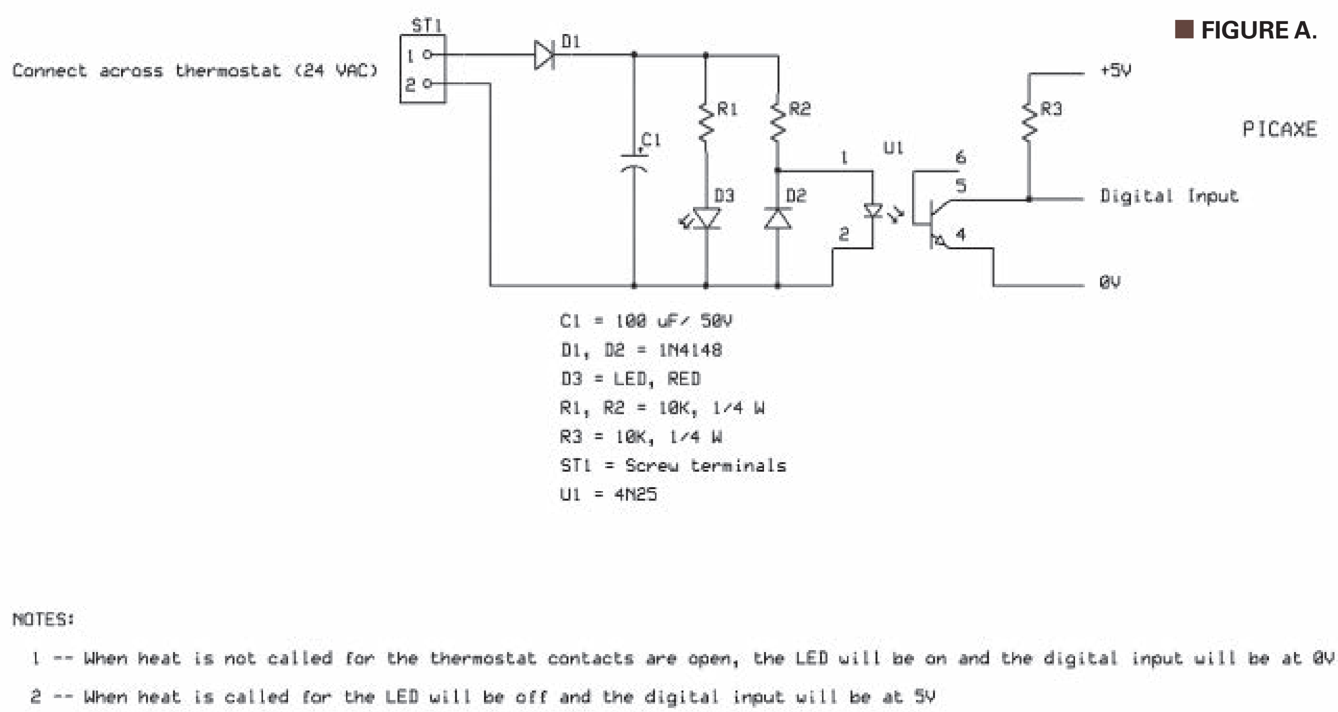
Charlie Heath

*Charlie, your idea sounds like a good way to monitor a furnace. I am including your schematic as **Figure A** so our readers can try it out.*

Re: Breaker Tripped/Sump Pump Alarm

Like some of your other readers, I noticed the schematic error in your December 2015 issue, but dismissed it as a drafting error. I expected a correction in a subsequent issue, and noticed the comments in the March 2016 edition. The proposed scheme is elegantly simple, but I believe serious issues remain. The first is that the alarm device is electrically conductive; it provides a path for current to flow downstream of the circuit breaker even if the breaker is open. If the circuit breaker is open and the sump pump is disconnected, say for repairs, there is potential for a serious electrical shock, with current limited only by the resistance/reactance of the alarm device.

For example, if a 120 VAC relay was employed as you suggest, the coil would still be connected between the "hot" bus and the sump pump line terminal. I measured the DC resistance of a small plug-in Potter &



■ FIGURE A.

Brumfield 120 VAC relay at about 2,100 ohms. If the breaker is open and the pump is disconnected, almost 60 mA could flow through the branch circuit conductors to result in a serious electrical shock -- especially to someone with wet hands and on wet concrete around a flooded sump; 60 mA could cause heart fibrillation and possibly death.

The second issue is the potential for fire. A substantial percentage of US structure fires are attributed to electrical distribution equipment causes. The circuit breaker is intended (and required) to prevent overheating of the downstream wiring. Your suggestion that the alarm device be connected to the "hot" bus in a breaker box essentially bypasses this important safety device. The "hot" bus is connected (through the service-entrance conductors and a disconnecting switch) to the electric utility transformer. Short circuit currents between the hot and neutral busses can be several thousand amperes, and branch circuit breakers are rated to interrupt 10,000A or more.

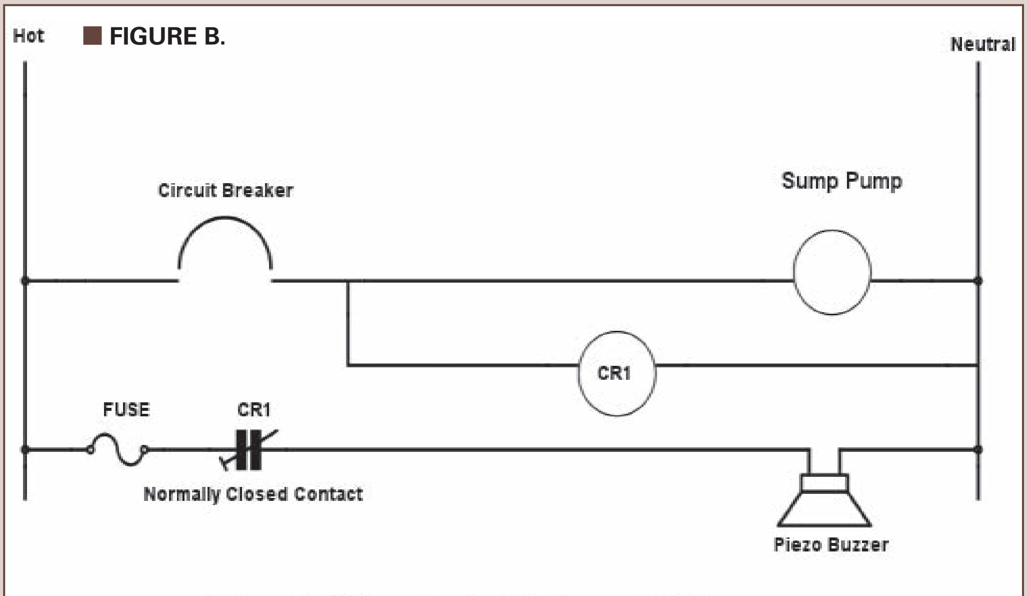
If the alarm device in your scheme was remotely located, the unprotected interconnecting cable would be a serious safety hazard and probably illegal in areas where electrical codes have been adopted. I have two alternative suggestions:

(A) If the sump pump is plugged in, a 120 VAC relay with NC or Form C contacts could be mounted in a suitable enclosure, and its coil connected through a grounded cord set plugged into an outlet on the sump pump circuit. The contacts could be wired to activate the alarm.

(B) The simplest safe solution might be to install a small low voltage float switch (less that $10 from **Amazon.com**) in the sump with NO contacts connected to the alarm. That would provide an alarm when the sump pump failed -- whatever the cause. I would go for a piezo electric beeper ($3-4 from All Electronics) powered by suitable alkaline batteries. The alarm and batteries could be installed where convenient, and connected to the float switch with low voltage bell wire.

David Toalson

■ FIGURE B.



Schematic of Circuit Breaker Trip Alarm with Relay

*David, your safety concerns are correct. I have included **Figure B** showing a relay (CR1) connected on the load side of the circuit breaker and its Normally Closed contacts wired from the "hot" power line through a piezo buzzer (or other suitable alarm). Sorry about using electrician's line diagram symbols and a ladder diagram, but in this case, it is warranted. This circuit should address all of your concerns about safety and fire. Moral of this story: Trying to simplify too much can be dangerous.*

Q&A SIDELINES

Modem Commands

Modem AT Commands

www.multitech.com/manuals/s000272j.pdf
www.perle.com/support_services/documentation_pdfs/5500158.pdf
www.zoomtel.com/techsupport/dial_up/1048pr3a.html#RTFToC102

Caller ID Software

www.ascendis.com/callerid/callerid.phpwww.callclerk.com
www.identafone.com/pro.html

BY L. PAUL VERHAGE
paul@nearsys.com

Supporting Near Space Missions with Quadcopters

One of the fastest growing "toys" in recent years is the quadcopter. These four-bladed flying machines are demonstrating amazing abilities to support a multitude of STEM (science, technology, engineering, and mathematics) related tasks — they're not just toys anymore. Being interested in high altitude science, I thought I'd begin investigating how a quadcopter might become a tool for serious amateur science and technology (when I wasn't having fun just playing with them, that is). This month, I'll introduce readers to how quadcopters can be helpful in the development of BalloonSats.

My Blade Chroma preparing to test a BalloonSat payload.

Quadcopters

Flying toys can make great tools for STEM education. They can inspire young people to become the STEM professionals (after graduation or even after earning a technical certification) who keep the world functioning, or even improve it to new levels that their elders would scarcely imagine. I believe BalloonSats represent one such activity. However, there's a new kid on the block that's growing like wildfire: quadcopters.

We frequently refer to quadcopters and its brethren like hexacopters and such as drones. These are multi-rotor devices that use GPS receivers, accelerometers, and magnetic compasses to stabilize their flights in ways that make them exceedingly easy to fly.

With their multiple rotors, they can carry a surprisingly large payload, and since modern technology can pack large capability into small packages, there's a lot of science and engineering that enthusiasts can do with a modern electric-powered drone. Of course, quadcopters can't reach near space altitudes; the air is just too thin. However, quadcopters can support the near space exploration in several ways.

BalloonSat, Meet Quadcopter

I started the process of teaching myself to fly my Blade Chroma quadcopter back in January of this year. The RC transmitter I purchased with the quadcopter was the Spektrum DXe: an inexpensive Bind-N-Fly (BNF) transmitter. Bind-N-Fly means I can use any Spektrum 2.4 GHz RC transmitter to fly any BNF-ready aircraft. So, purchasing one BNF transmitter lets me fly any future model aircraft using this transmitter.

This will save me money since I won't need to purchase a new transmitter every time I purchase an upgraded drone or any new model aircraft.

I purchased the Chroma when I discovered that it was designed to carry a GoPro camera. Online references tell me that the basic GoPro weighs in at around seven ounces, meaning that any near space payload being tested onboard a drone can't weigh much more than that. That's actually pretty close to the weight of BalloonSat avionics anyways, so there's not a problem here. Since the quadcopter is inexpensive to fly relative to a weather balloon, aerial testing payloads on a quadcopter makes sense as a way to debug BalloonSat avionics.

However, what makes the quadcopter an even better test bed is how much fun and excitement there is in flying one (the eighth grader inside me was very excited to watch my quadcopter take off with the avionics I was testing). I suppose in some ways, testing BalloonSat avionics in a quadcopter is like NASA testing satellite avionics in near space onboard a weather balloon.

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/June2016_Near-Space-Missions-Meet-Quadcopters**.

My Quadcopter Aerospace Test Bed

On my first scientific quadcopter flight, I tested a new air temperature and air pressure sensor that I was developing. If it worked well on this test, then I'd plan to test it on a near space flight, and then finally market kits on my website (**NearSys.com**). I can run tests on my bench all I want, but nothing beats collecting some real data in flight. However, first I had to design a payload carrier for the Chroma. There's a camera mounting rail built into the underside of the Chroma for a GoPro camera that I adopted specifically for the payload carrier.

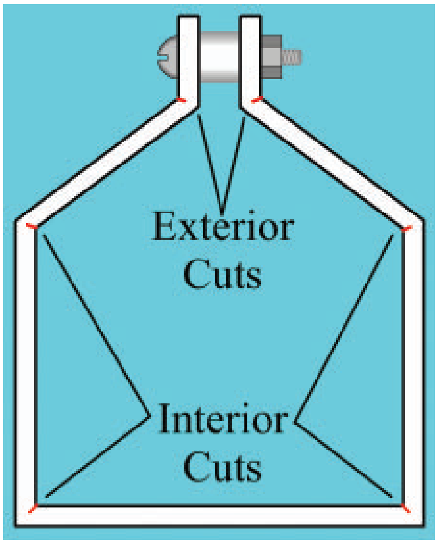
The Chroma is designed to carry camera payloads for some pretty awesome aerial footage. In this case, the Chroma is carrying science sensors to altitude.

I made the payload carrier from Coroplast: a strong, lightweight, 3/16" thick corrugated plastic. Unlike near space missions where intense cold is an issue, quadcopter flights never get colder than a few degrees lower than ground temperature. So, I decided that Coroplast (as opposed to Styrofoam) would make an excellent payload harness.

Because of its thickness, Coroplast can be difficult to bend. Therefore, I cut through one wall of the channel where I wanted to bend the plastic. After cutting and bending, I ended up with a house-shaped structure that I could bolt to the GoPro clamp in the quadcopter's camera mounting rail.

I cut and drilled holes in the payload harness so I could attach the avionics and camera to it, and so that the camera lens could see outside the harness. In this payload carrier design, the camera is bolted inside the payload carrier and the avionics are bolted to the exterior.

Above the camera (but still inside the harness) is a block of foam rubber and battery for the avionics. The foam rubber has two functions. First, it fills the empty volume of the interior, and second, it holds the battery in place so it can't fall out of the carrier.



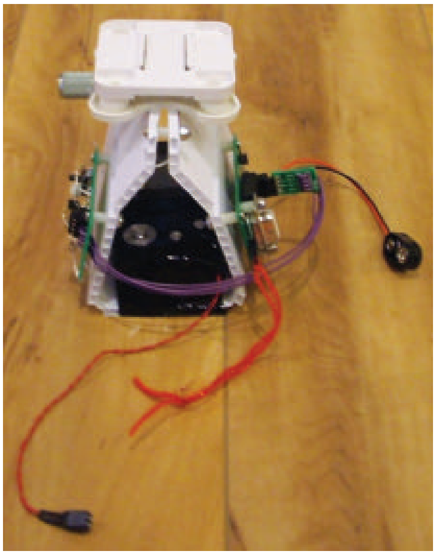
The locations of the cuts in the Coroplast are shown in red. After cutting the interior sides of each bend, the Coroplast sheet was folded into the house shape shown here. The gap and spacer at the top of the harness is necessary to make it fit within two slots provided for the GoPro camera clamp.

Performance and Limitations

After completing the payload harness and attaching it to the Chroma drone, I set the quadcopter down in the front yard, powered it and the transmitter up, and waited while the quadcopter got a GPS fix. In my admittedly less than precise test, I observed my Chroma carry its eight ounce payload to an altitude of 200 feet in about 25 seconds. That's an ascent rate of eight feet per second or about half the ascent rate of an average near space launch.

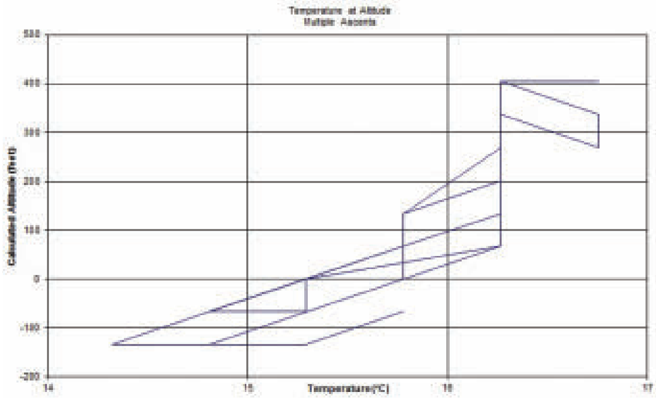
How did I estimate the altitude? I did so by using ratios and geometry. The thumb at the end of my outstretched arm is 28 inches away from my eye. The Chroma is one foot across, and I wanted it to hover at an altitude of 200 feet. Therefore, I created a ratio between the hover altitude of the Chroma and its diameter, and equated that ratio to the ratio of arm length to distance across my thumb. So, in other words, mathematically:

$$\frac{\text{Width across Thumb (my unknown)}}{\text{Arm Length (28 inches)}} = \frac{\text{Width of Quadcopter (one foot)}}{\text{Altitude (200 feet)}}$$

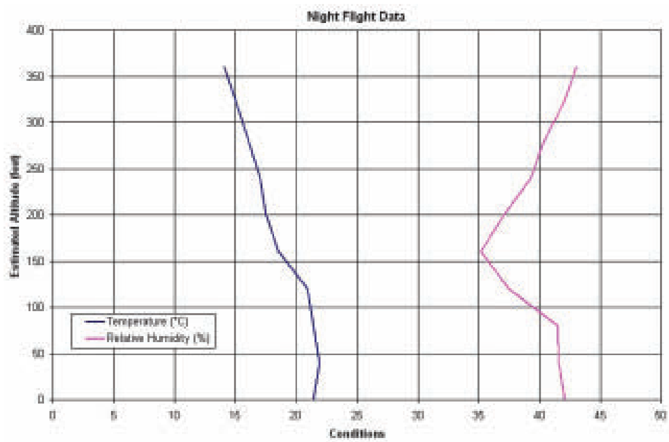


The payload harness loaded up with BalloonSat avionics. At the top of the harness is a GoPro mounting bracket that attaches my payload harness (or a GoPro) to the Chroma's camera rail.

You can find a variation of the pressure-altitude equation at the SparkFun website. I changed the equation to convert the altitude from meters to feet and the sea level pressure into units of millibars. I then modified the exponent to a number rather than use a fraction. See **https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/measuring-weather-and-altitude** for the equation.



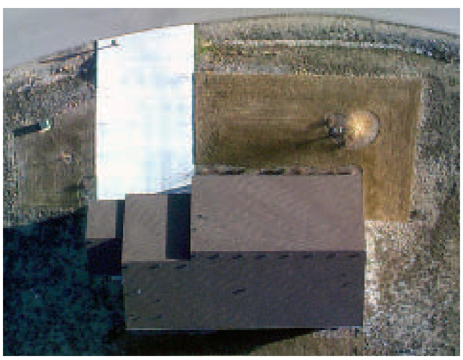
The graph of temperature and air pressure collected during a daylight flight. There's a spread in the data since the temperature sensor wasn't given enough time to come to temperature at each altitude.



This data was collected a few days after the temperature and pressure data, and represents a more logical approach to data collection.

Using algebra to solve for the variable (which is *Width across Thumb*) yielded an answer of 0.14 inches. Therefore, when the quadcopter reaches an altitude of 200 feet, it would appear to span a width of 0.14 inches across my thumb when my arm was outstretched. To make the measurement practical, I drew two marks roughly 0.14 inches apart on my thumb before going outside to launch the quadcopter. When I launched the quadcopter, I kept the throttle set high and monitored the apparent width of the quadcopter as it climbed. When it spanned the 0.14 inch marking on my thumb, I pulled back on the throttle and set the quadcopter to hover.

I let the quadcopter hover for about a minute at an altitude of 200 feet before bringing it down and repeating the ascent, climbing to an altitude closer to 300 feet (based on the apparent size of the quadcopter). All the while, the quadcopter's payload was collecting data and taking photos.



This is what my house looks like from an altitude near 300 feet.

The flight computer on board the quadcopter recorded data at 10-bit resolution every 10 seconds. So, after landing the quadcopter, I connected the flight computer to a PC and downloaded its temperature and pressure data. I used the pressure data to calculate an estimated altitude for each temperature reading. The equation I used to calculate altitude from pressure is highlighted below. I didn't use the exact altitude calculated by the equation since it doesn't account for the air pressure at my house, nor the air temperature, relative humidity, or elevation. Instead, I subtracted the calculated altitudes from the altitude at the start of the flight. That let me calculate the change in the quadcopter's altitude, or its altitude above my house. That's good enough for a test like this.

There is an inaccuracy in the temperature readings since I didn't let the quadcopter (and therefore the LM335 temperature sensor) hover at fixed altitudes long enough for the temperature sensor to reach air temperature. I flew the quadcopter up to attitude several times and graphed the results.

For the second science flight, I launched the quadcopter at night and hovered it for 10 seconds at regular intervals. In this case, the quadcopter climbed for five seconds at full throttle (so, around a 40 foot change in altitude each time), and then hovered long enough to collect a temperature and relative humidity measurement. I repeated this 10 times and collected data up to an altitude of around 360 feet (within the limits of precision this method permits). I later learned the maximum altitude for the Chroma is 100 meters, or 328 feet. So, my estimated altitude wasn't too far off.

Pretty cool! I'm now convinced the quadcopter can support BalloonSat building and testing by letting students test their avionics on a quadcopter flight. It would even be better (in my opinion) if the students can help fly the quads. Quadcopters, I believe, can do more than just test avionics for BalloonSats. Next time, I'll report on my investigation on using the quadcopter at near space launches and recoveries.

Onwards and Upwards,
Your near space guide **NV**

$$\text{Altitude (feet)} = (1 - [\text{Pressure (mb)}/1013.25]^{0.190284}) * 145366.45$$

NEW PRODUCTS

- HARDWARE
- SOFTWARE
- GADGETS
- TOOLS

3/8" SERVO **SHAFTS**

Two new patented servo shafts from Actobotics® are extremely useful for a variety of applications, turning a servo spline into a functional 3/8" output shaft. Machined from 7075 aluminum, the units are very strong, and are offered in both 24T and 25T splines.

Generally, the 24T versions will work with Hitec servos, and the 25T versions with Futaba. However, many of the new Hitec servos have a 25T spline, so be sure to double-check your spline size before ordering. Available in 0.280" and 1.00" lengths (the 0.280" versions are designed specifically to work with the ServoCity clamping shaft couplers), pricing ranges from $3.99-$5.99

RUNT ROVER **ZIP KIT**

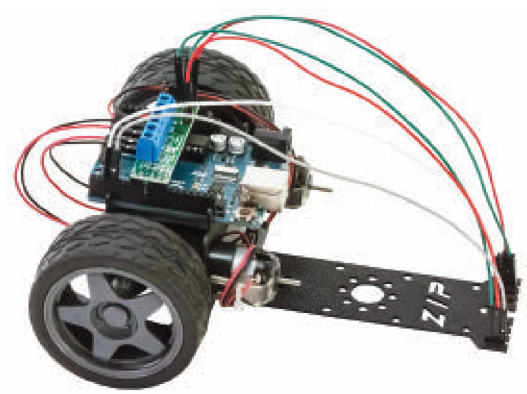
ServoCity is now offering the Zip Kit: the latest Runt Rover™ to join the group. The Zip includes all the components and code needed to build a zippy little line following robot, making it perfect for getting started in robotics and programming. This plug-and-play kit includes an Arduino compatible board preloaded with a line following sketch. Simply download and modify the commented sketch to adjust the performance.

It easily snaps together. The only tool required is a small slot headed screwdriver for the screw terminals. This kit is powered with a 9V battery (included).

Other included parts are:

(2) 140 RPM right angle gear motors
(2) 2.55" press fit wheels
(1) Arduino Uno R3 compatible development board
(1) Dual motor driver shield for Arduino
Jumper wires
  (1) Red, M-F (1) Gray, M-F (2) Green, M-F
  (1) Red, F-F (1) White, M-F
(1) 1x4 stackable header row
(1) 9V connector
(1) 9V battery
(2) QTR-1RC reflectance sensors
(1) ABS body

A wiring diagram, explanation of the code, and instructional videos for assembly guidance are all available. Price is $49.99.



For more information, contact:
ServoCity
www.servocity.com

If you have a new product that you would like us to run in our New Products section, please email a short description (300-500 words) and a photo of your product to:

newproducts@ nutsvolts.com

DAQami COMPANION SOFTWARE

Measurement Computing Corporation (MCC) has announced the release of DAQami version 3.0. DAQami is an easy-to-use data acquisition application compatible with most devices from MCC.

New features in version 3.0 include support for analog, digital, and counter/timer outputs. Users can now add the following functions to their DAQami application: set wave type, frequency, and duty cycle of analog outputs; control digital outputs on a per channel basis; and set frequency and duty cycle of counter/timer outputs.

DAQami also allows users to display and log data, review data, and export data to programs like Microsoft® Excel or MATLAB®. Users can quickly configure acquisition and channel options and choose from a variety of data displays.

DAQami can be downloaded from **www.mccdaq.com/DAQami** and customers can try the fully-functional software for 30 days — including data acquisition, data logging and export, and signal generation capabilities. After the initial 30 days, all features except for data logging and data export will continue to be available. Users can unlock data logging and data export features after the initial 30 days for only $49. DAQami is supported under Microsoft Windows 10/8/7/Vista® operating systems.

For more information, contact:
Measurement Computing
www.mccdaq.com

LIGHTING FLICKER EFFECTS CONTROL

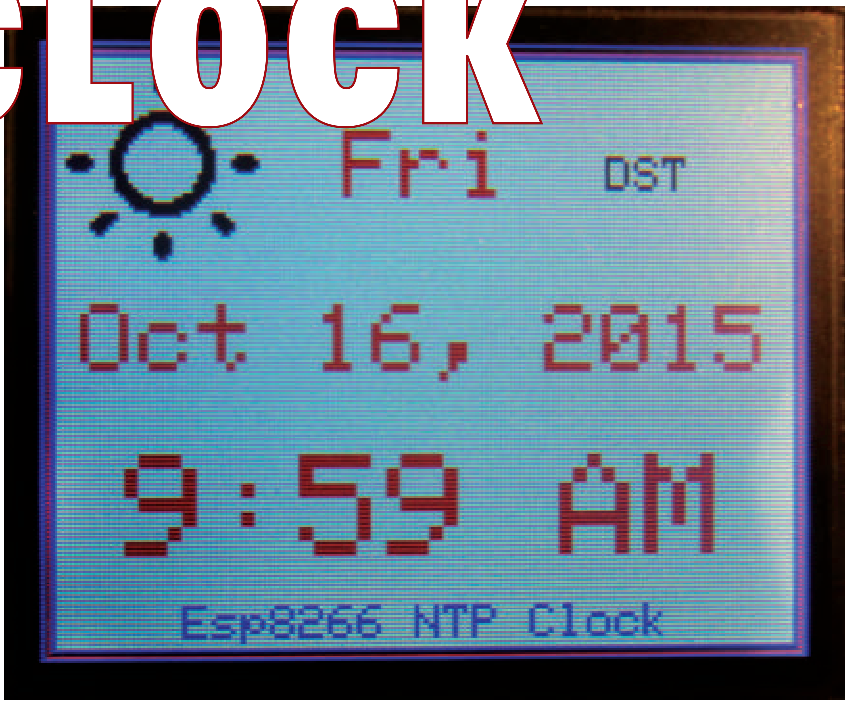
The J2 LED Lighting Flicker Effects Control (FEC) has six dynamic effects modes and one static dimming mode. The FEC is intended for simulation of flame and ember sources such as candles, lanterns, oil lamps, fires, coals, and wood embers. Each of the six modes is distinguished by the average light level, the maximum depth of the flicker trough, and a different flicker pattern.

Modes with the softest flicker appearance have lower flicker trough depth (to 70%). Modes of greatest flicker appearance will have greater flicker troughs, 80% to 90% maximum. All dynamic modes have 10 speed steps: step #1 fast (flame

Continued on page 61

ESP8266 NTP CLOCK

By Craig A. Lindley



Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/June2016_ESP8266-NTP-Clock.**

Building digital clocks is not the sexiest of DIY projects, yet many people do so each year. People build these clocks in a wide variety of shapes and sizes, including the weird one I designed and wrote about in the March 2014 issue called, "A Unique LED Clock." Most homebrewed digital clocks use an Arduino or other microcontroller coupled to a real time clock (RTC) chip that provides the time keeping machinery and (in some cases) battery backup facilities. It is up to the user to set the clock to the correct time. If good quality components are used in the clock, time keeping accuracy can be pretty good. However, unless the RTC chip's oscillator is temperature controlled, accuracy will drift over time forcing the user to perform periodic corrections. Also, some RTC chips don't handle daylight saving time (DST), so it is up to the user to reset their clocks twice a year in affected areas.

To overcome the problems with manual time, date setting, and time drift, many so called "atomic clocks" or "radio controlled clocks" have appeared on the market. They come in just about every conceivable shape and size. These clocks listen for WWV radio transmissions from Fort Collins, CO, and synchronize their time keeping mechanisms to the atomic clock reference used for these transmissions. This helps guarantee their time keeping accuracy. Clocks like these typically require the user to select their time zone, but other than that do not offer any controls for manually setting the time.

The clock mechanisms in PCs work differently. PCs usually sync their RTC using an Internet standard called Network Time Protocol, or NTP.

According to Wikipedia:

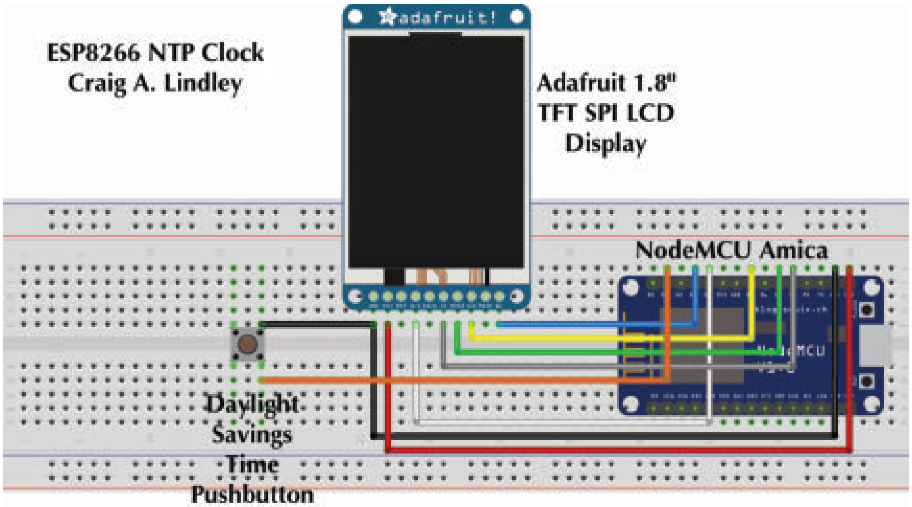
NTP is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.

NTP is intended to synchronize all participating computers to within a few milliseconds of Coordinated Universal Time (UTC).

NTP can usually maintain time to within tens of milliseconds over the public Internet, and can achieve better than one millisecond accuracy in local area networks under ideal conditions.

Basing a digital clock design on NTP requires access to the Internet which can be expensive to implement, but allows for a very simple clock design for a couple of reasons. First, no battery backup circuitry is required to maintain the time setting. If clock power is lost, the connection to the Internet will automatically be reestablished once power is restored; the clock will automatically set itself to the correct time. Second, no controls for manually setting the time are typically necessary because time and date settings are automatic.

The ESP8266 family of devices makes inexpensive access to the Internet a non-issue, so it's natural to use these devices in an NTP clock. Current readers of *Nuts & Volts* may remember my two previous articles about using the amazing ESP8266 devices: "Meet the ESP8266: A Tiny, Wi-Fi Enabled, Arduino Compatible Microcontroller" in the October 2015 issue; and "Thinking of You" in the November 2015 magazine.



■ **FIGURE 1. ESP8266 NTP clock wiring diagram/schematic.**

To refresh your memory, all members of the ESP8266 device family share some basic characteristics, including:

- 802.11 b/g/n
- Wi-Fi Direct (P2P), soft-AP
- Built-in TCP/IP protocol stack
- 802.11b mode + 19.5 dBm output power
- Built-in temperature sensor
- Supports antenna diversity
- Off leakage current is less than 10 µA
- Built-in low power 32-bit CPU which can double as an application processor
- SDIO 2.0, SPI, UART, ADC, EEPROM
- Standby power consumption of less than 1.0 mW (DTIM3)

In other words, the ESP8266 family of modules features low power consumption, high RF power output, and are capable of supporting all of the current 802.11 standards required for Wi-Fi connectivity. In addition, they support many industry standard hardware interfaces and can function as the application processor in many designs as they do in this one. The ESP8266 is a 3.3 VDC part.

Two things make using these parts even sweeter. First, many ESP8266 modules can be purchased for under $10 in single unit quantities. Second, these modules can be programmed in the Arduino IDE (integrated development environment), so Arduino developers don't have to learn yet another programming system to use them.

In this article, I present the design and implementation of a very simple NTP digital clock based on the ESP8266 that drives a small LCD display. In actuality, I used an ESP8266 variant called a NodeMCU LUA Amica as it has lots of digital I/O pins available, making interfacing to the display trivial. This clock has a single pushbutton switch that — if configured for daylight saving time operation (more on this later) — allows the user to put the clock into and take the clock out of DST mode.

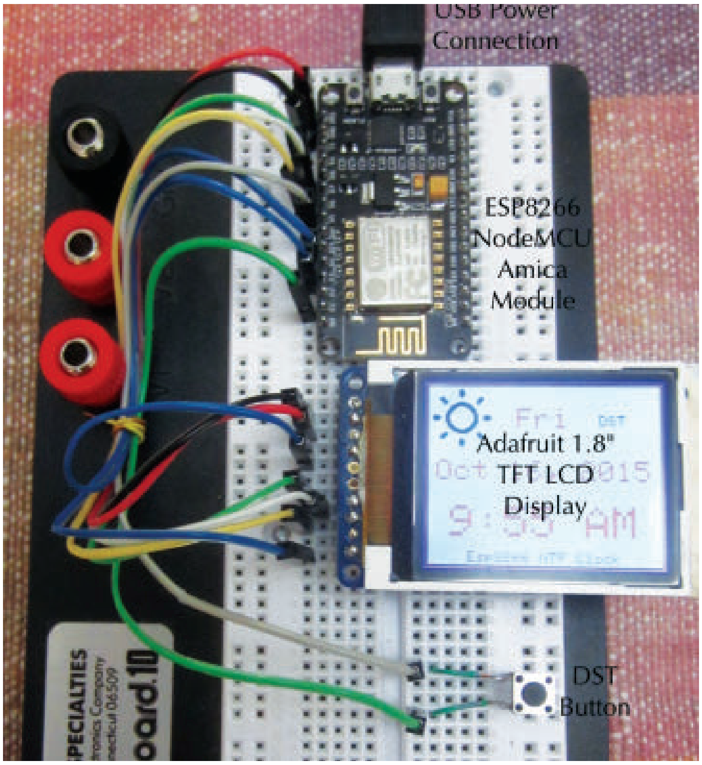
Designing this digital clock allowed me to experiment with aspects of the ESP8266 that I had not used before, including the hardware SPI interface used to run the LCD display and the onboard EEPROM for storage and retrieval of the DST state indicator.

Hardware

The hardware **Parts List** shows the items required to build one of these NTP clocks and where to get them. As you'll see, there isn't much to it. **Figure 1** shows a Fritzing connection diagram/schematic for the NTP clock. **Figure 2** shows the design wired up and working on a breadboard. NOTE: There isn't a wire color correlation between

PARTS LIST

PART	SOURCE
NodeMCU LUA Amica R2 Module	**Electrodragon.com**
1.8" TFT SPI LCD Display	**Adafruit.com** — Product ID: 358
Pushbutton Switch SPST	Your favorite electronics store
USB Cable — USB A to USB Micro B	Your favorite electronics store
USB Power Supply (capable of at least one amp at five volts)	Your favorite electronics store



■ **FIGURE 2. The ESP8266 NTP clock breadboard.**

Figures 1 and **2**. As shown, the clock is powered via a USB cable and a USB power supply module. The wire by wire connections are shown in **Table 1** (in case they're not be clear from the Fritzing diagram). The GPIO designations are also shown in **Table 1**. This is how these digital I/O lines are referred to in the Arduino code.

The Adafruit LCD display has a microSD memory card connector and interface which can be used with the ESP8266; these were not needed for this particular project.

Software

The software for the ESP8266 NTP clock was developed using the Arduino IDE. Refer to my previous articles or the **Resources** section for how to set up the Arduino IDE on your computer for targeting ESP8266 type devices. Make sure to select *"NodeMCU 1.0 (ESP-12E Module)"* as the board type in the Tools menu.

The ESP8266 NTP clock software is available at the article link; it's called *Lindley_ESP8266NTPClock.zip*. To use the software, unzip it and copy/move the ESP8266NTPClock directory from the zip file into your Arduino directory.

Whereas the hardware for this clock borders on the trivial, the software/firmware for the clock is a bit more involved and complex. The seven files which make up the code are described in **Table 2**.

The ESP8266_ST7735 LCD driver code was adapted from the Adafruit ST7735 library to use the hardware SPI interface on the ESP8266. If you want to use a different LCD display, you will have to find/develop an appropriate driver yourself.

In addition to the files above, the Arduino libraries in **Table 3** are also required.

The version of these libraries I used to develop the NTP clock is included in the zip file for this article. Remember, libraries must be installed in the *arduino/libraries* directory on your development computer and the Arduino IDE must be restarted to recognize them.

Most of the code that makes up the NTP clock is straightforward and will be easy to understand. The NTP code in the file *NTP.h* is more complex, however. To retrieve the time, one must get an IP address of a time server from the pool of *time.nist.gov* servers using the *hostByName* function as shown in the following code. If you were to monitor the *timeServerIP* address during the operation of the clock, you would see that the time requests rotate in a round robin fashion between the servers in the *time.nist.gov* pool:

```
// Get a server from the pool
WiFi.hostByName("time.nist.gov", timeServerIP);
```

Once you identify a server to make a request to, you must create a UDP packet configured with the proper values and then send it the packet. See **https://tools.ietf.org/html/rfc5905#section-7.3** for an explanation of the fields in the UDP request packet. The following *sendNTPPacket* function does this:

```
// Send an NTP request to the time server
//  at the given address
unsigned long sendNTPpacket(IPAddress&
address) {

  // Set all bytes in the buffer to 0
  memset(packetBuffer, 0,
NTP_PACKET_SIZE);

  // Initialize values needed to form NTP
  // request
```

Table 1.

NodeMCU Amica Pin	Adafruit 1.8" Display Connection	DST Pushbutton SPST Switch
D1 (GPIO 5)		SW1
D3 (GPIO 0)	LITE	
D4 (GPIO 2)	D/C	
D5	SCK	
D7	MOSI	
D8 (GPIO 15)	TFT_CS	
3V3	VCC	
GND	Gnd	SW2

```
  packetBuffer[0] = 0b11100011;
   // LI, Version, Mode
  packetBuffer[1] = 0;
   // Stratum, or type of clock
  packetBuffer[2] = 6;
   // Polling Interval
  packetBuffer[3] = 0xEC;
   // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root
  // Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // All NTP fields have been given values, now
  // you can send a packet requesting a
  // timestamp:
  udp.beginPacket(address, 123);
    // NTP requests are to port 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
}
```

Once the UDP packet is sent, you wait for a response; in that response will be a time stamp (the four bytes starting at the 40th byte of the response) indicating the time the packet was sent. The units of this time stamp are seconds (since 1900) and it is a very large number. This value gets converted to Unix time (which is seconds since January 1, 1970) by the subtraction of the number of seconds between 1900 and 1970. This number is then further modified via time zone correction. This corrected value is used by the Time library, and converted to the current time and date displayed by this clock. The *getNTPTime* function pulls this all together:

```
// NTP Time Provider Code
time_t getNTPTime() {

  int attempts = 10;

  // Try multiple attempts to return the NTP
  // time
  while (attempts—) {

    // Get a server from the pool
    WiFi.hostByName(ntpServerName,
    timeServerIP);
    Serial.print("Time server IP address: ");
    Serial.println(timeServerIP);

    while (udp.parsePacket() > 0);
    // Discard any previously received packets

    Serial.println("Transmitted NTP Request");
    sendNTPpacket(timeServerIP);

    uint32_t beginWait = millis();
    while (millis() - beginWait < 1500) {
      int size = udp.parsePacket();
      if (size >= NTP_PACKET_SIZE) {
        Serial.println("Received NTP
        Response");
        udp.read(packetBuffer, NTP_PACKET
        _SIZE);
        // Read packet into the buffer
        unsigned long secsSince1900;

        // Convert four bytes starting at
        // location 40 to a long integer
        secsSince1900 =  (unsigned long)
        packetBuffer[40] << 24;
        secsSince1900 |= (unsigned long)
        packetBuffer[41] << 16;
        secsSince1900 |= (unsigned long)
        packetBuffer[42] << 8;
        secsSince1900 |= (unsigned long)
        packetBuffer[43];

        Serial.println("Got the time");

        return secsSince1900 - 2208988800UL +
        realTimeZoneOffset * SECS_PER_HOUR;
      }
      delay(10);
    }
    Serial.println("Retrying NTP request");
    delay(4000);
  }
  Serial.println("No NTP Response");
  return 0;
}
```

Table 2.

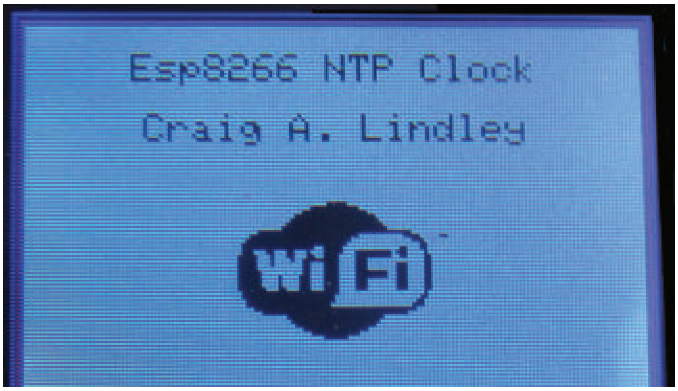
File	Description
ESP8266NTPClock.ino	Main program. Initializes the hardware, logs into the local Wi-Fi network, and then installs the NTP code as the time provider. It then manages the update of the clock on the display.
ESP8266_ST7735.cpp	LCD driver code specific to the Adafruit 1.8" (black tab) display utilizing the hardware SPI interface of the ESP8266.
ESP8266_ST7735.h	Header file for the LCD driver code above.
Icons.h	Data for the Wi-Fi, sun, and moon icons. Data is in xbm format.
NTP.h	Functions for sending UDP packets to NTP servers, and retrieving the GMT time and converting it to local time.
TextGraphicsFunctions.h	Misc functions for formatting the time data for display on the LCD.
Misc.h	Code for reading and writing the ESP8266's EEPROM.

User Configuration of the NTP Clock Software

The NTP clock's software must be configured before the clock will work correctly. All user configuration items are found in the *ESP8266NTPClock.ino* file. Please locate the following code in that file:

Table 3.

Library	Source
Adafruit_GFX	https://github.com/adafruit/Adafruit-GFX-Library
Time	https://github.com/PaulStoffregen/Time



■ **FIGURE 3. Initial Wi-Fi connection display.**

```
// ********************************************
// Start of user configuration items
// ********************************************

// Set your WiFi login credentials
#define WIFI_SSID "xxxxxxxx"
#define WIFI_PASS "xxxxxxxxxxxxx"

#define TIMEZONE_OFFSET -7
   // Set your timezone offset (-7 is mountain
   // time)
#define USE_DST          true
   // Set to false to disable DST mode
#define HOUR_FORMAT_12  true
   // Set to false for 24 hour time mode

// ********************************************
// End of user configuration items
// ********************************************
```

First — and most importantly — you must modify the code with the SSID and password of your Wi-Fi network. Otherwise, the clock won't be able to access the Internet, and by extension the NTP servers that provide the time. Next, you must set the correct time zone offset for your location. Time zone offsets can be found at **https://en.wikipedia.org/wiki/List_of_UTC_time_offsets**.

You must then decide if your clock will use DST or not, and whether it will operate in 12 or 24 hour format. *USE_DST* must be set true if your clock will use daylight saving time, whether or not DST is currently in effect. Set *HOUR_FORMAT_12* true to run your clock in 12 hour format; otherwise, it will operate in the 24 hour time format.

The code can be compiled and uploaded to the NodeMCU device once the configuration data is set and all of the required libraries have been installed in the Arduino environment.

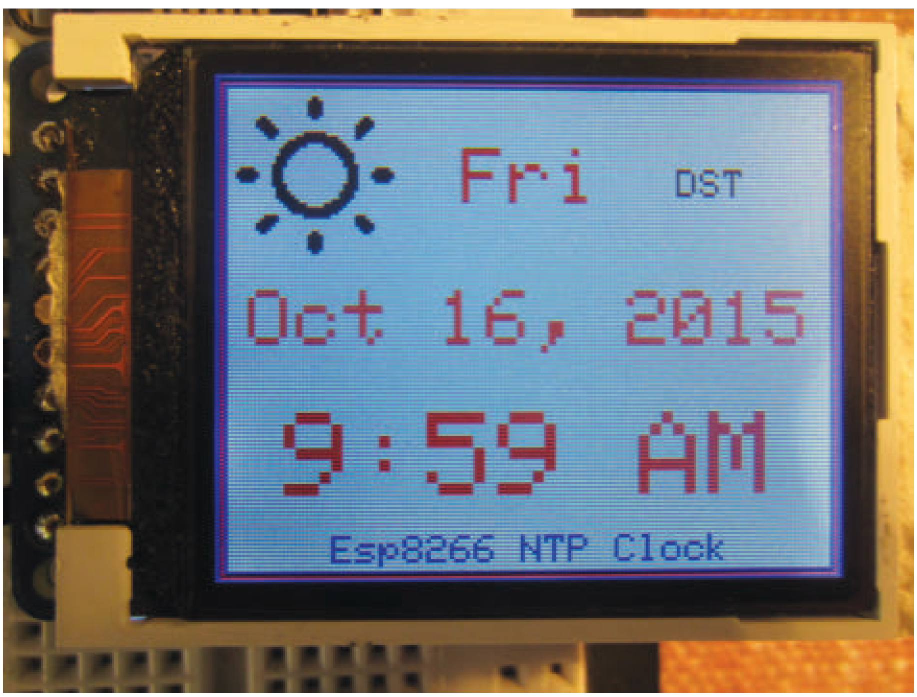
NTP Clock Operation

The clock should start immediately once the software is uploaded. **Figure 3** shows the clock's display while a connection is being made to the local Wi-Fi network. If this screen doesn't change to the clock display of **Figure 4**, it means there were problems logging into the Wi-Fi network. If this is the case, go back and verify the *WIFI_SSID* and *WIFI_PASS* entries in the code, and that the Wi-Fi network is working.

As mentioned, the Wi-Fi login display should change to the clock display in **Figure 4** once a Wi-Fi connection is established. If the clock is not configured for DST mode, you are done. The clock should run as long as power is applied, and it will sync up its time to an NTP time server every five minutes. This makes the clock extremely accurate.

If the clock is configured for DST operation (*USE_DST* is true), it is up to the user to put the clock in DST mode if DST is currently in effect (mid April through November in the US). The clock doesn't default to DST mode, so the user must push the DST button until the DST string is displayed in the upper right corner of the clock. You'll notice the displayed time changes when DST mode is engaged. Pressing the DST button again toggles the clock out of DST mode.

The clock will continue to run as long as power is applied. If the Internet connection is dropped, the clock will



■ **FIGURE 4. Typical clock display. Note: Daylight saving time (DST) mode is on; when the sun icon is displayed, it is daytime.**

maintain the time itself. If Wi-Fi goes down but the clock remains powered, the clock will need to be rebooted once the network issue is resolved so NTP time syncing can be restarted. If power is lost to both the clock and the Wi-Fi network, the clock will reboot and wait for the network to come back up, and then will reconnect automatically.

While the clock is operational, the time and date will update once a minute. During the day, a sun icon will be displayed in the upper left corner. The sun icon will be replaced by a moon icon after about 8 p.m. in the evening.

As daylight saving time comes and goes, the user will be required to inform the clock by pressing the DST pushbutton which toggles this mode on and off. Other than that, there are no other ongoing operational maintenance issues required by the user.

As a final note, the DST enabled state is written to the EEPROM in the ESP8266 every time the DST pushbutton is pressed. This was necessary to bring back the correct DST state if power to the clock was lost and then regained. See the file *Misc.h* for the EEPROM read/write code.

I hope you've enjoyed your time with me here, and consider building yet another clock of your own design. **NV**

Resources

The following resources may be of use:

Information about NTP can be found all over the Internet. See **www.ntp.org** for detailed data.

Information on WWV time broadcasts can be found at **https://en.wikipedia.org/wiki/WWV_(radio_station)**.

Information about programming the ESP8266 in the Arduino environment is available at **github.com/esp8266/Arduino** and in my two articles mentioned previously.

Information about the NodeMCU Amica can be found at **www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board**.

Information about the Adafruit 1.8" TFT SPI LCD display is available at **www.adafruit.com/products/358**.

A different version of this clock which uses NeoPixels instead of an LCD display can be found at **www.craigandheather.net/celeledclock2.html**

Medical Grade Bedwetting ALARM AND SENSOR

By Jeff Tregre

This step-by-step project is a complete bedwetting alarm and sensor that can be built for around $50. The alarm can be used for mainly three groups of people: children, the elderly, and people with diabetes. Some of the complete store bought systems sell for around $200, so this is a savings, and like all the projects in this magazine, you have the reward of building it yourself.

Now, it should be clear as to why both children and the elderly might have need of a bedwetting alarm, but why would a diabetic have a need for this device? This is because as their blood sugar drops below 60 due to insulin or oral medication, the body will begin to sweat. If this happens during the night while they are sleeping, they may wet the bed the size of their upper torso. If their blood sugar continues to drop while they are asleep (hypoglycemia), they could go into a coma and never wake up again. This project could possibly interrupt what might become a potentially deadly situation.

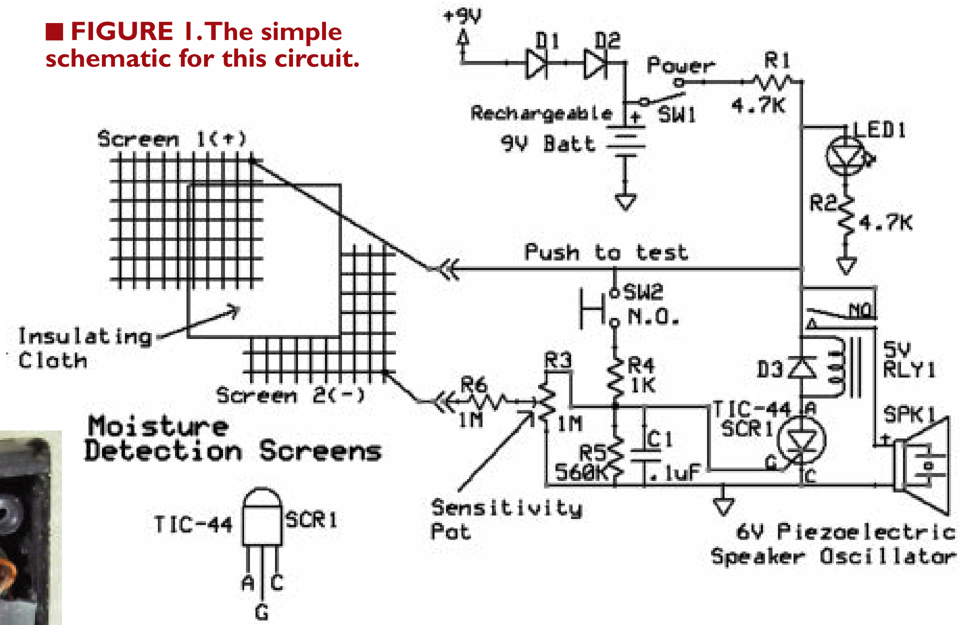
Theory of Operation

To power this circuit, I am using a +9 VDC wall adapter power supply. The voltage output is slightly higher than nine volts; therefore, I have installed two diodes — D1 and D2 — in series to help lower the voltage and help keep the 9V battery charged, but yet cool. The rechargeable 9V battery is there simply as a backup provision should power be lost during the night.

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/June2016_Bedwetting-Alarm-Sensor.**

Resistor R1 (4.7KΩ) is used to drop the 9V down to about 5.5V. R2 is another (4.7KΩ) resistor used to keep the blue LED lit, but dim so that it is not that noticeable at night with all the lights off.

About 5.5V goes to the top Screen 1 (+). It is insulated by a thin pillowcase and placed on top of Screen 2 (-). The normal resistance between the two screens when dry is several



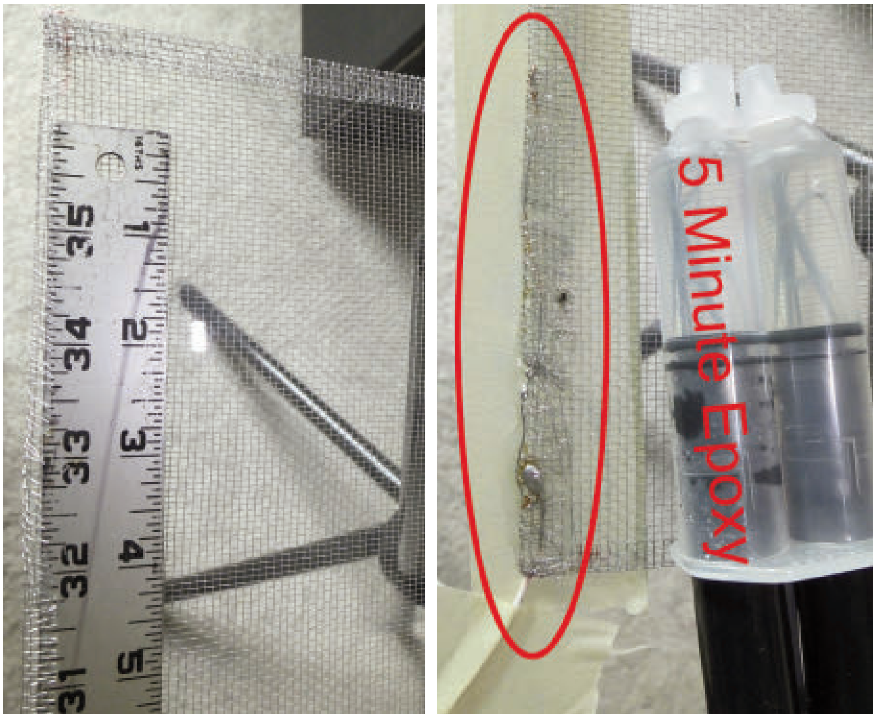
■ **FIGURE 1. The simple schematic for this circuit.**

■ **FIGURE 2. The finished unit inside.**

■ **FIGURE 3.**

PARTS LIST

ITEM	DESCRIPTION
+9 VDC wall adapter power supply	
D1, D2 & D3	Any general-purpose diodes will work
Rechargeable 9V battery	
R1 & R2	(4.7KΩ) resistors 1/4W
R3	1MΩ potentiometer
R4	1KΩ resistor 1/4W
R5	560KΩ resistor 1/4W
R6	1MΩ resistor 1/4W
C1	.1 µF capacitor
SCR1 — TIC-44	Any general-purpose SCR should work
LED1	Any color will work
Switch 1	Pushbutton SPST, push ON/push OFF
Switch 2	Pushbutton SPST, normally open
RLY1	5V relay, SPDT
SPK1	6V piezoelectric speaker
• Screens 1 and 2	Any fine mesh aluminum insect screening will work. The screen must be metallic or it will not conduct. The thinner, the better.
• Insulating cloth	I used two pillowcases 20" x 30" in size
• Connector jacks	You may wish to purchase two mating connector jacks: one for the power supply, and the other for the screen sensors.
• Perforated printed circuit board	Small size with copper pads on the bottom side.
• Plastic project case	Most any small size plastic case should work.

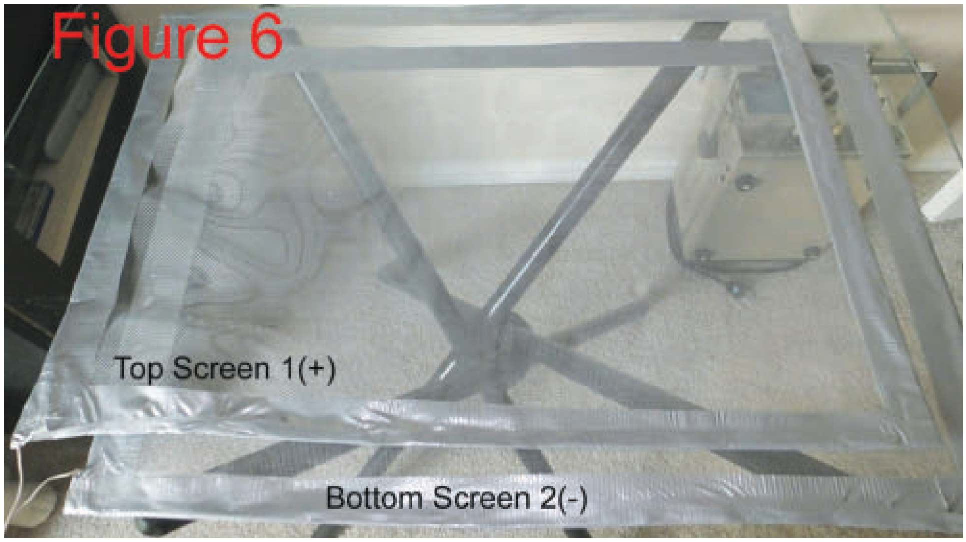


■ **FIGURES 4 and 5.**

megohms; however, when body fluid wets the insulating thin pillow case separating the two screens, the resistance between them drops to just a few hundred ohms. This, in turn, slowly charges capacitor C1 (.1 µF) and forward biases the gate on the silicon controlled rectifier (SCR1). When SCR1 is turned on, it provides a path to ground thereby allowing the 5V relay (RLY1) to turn on and stay on. When the relay is turned on, it provides power to the piezoelectric speaker (SPK1) which sounds and wakes you up letting you know moisture is being detected.

This circuit will work without the relay; however, as the forward biasing voltage to the gate of the SCR begins to increase, it begins to chatter the piezoelectric speaker. The relay helps to prevent this chattering.

I have added a push-to-test switch (SW2) to my circuit which simulates the screens becoming wet and sounds the piezoelectric speaker. To reset the circuit, simply turn off SW1 and then turn it back



■ **FIGURE 6.**

on again. R3 is a 1 MΩ potentiometer. This pot allows you to fine-tune the circuit based on temperature and humidity, as well as the fact that many people do perspire during the night — especially during the winter months when they are all wrapped up in blankets to keep warm. Therefore, R3 will allow you to fine-tune the circuit based on your own personal preference as to the mount of moisture the sensor triggers on.

Resistor R6 (1 MΩ) was added after the circuit was built and tested. This resistor helps to limit the sensitivity of the circuit so that it will not sound the alarm due to only small amounts of perspiration. If need be, you may wish to change the value of this resistor for your own personal preference. If the value of R6 is increased, the unit will become less sensitive. If the value of R6 is decreased, the sensitivity of the unit will increase.

Constructing the Moisture Detection Screens

The sensor screen can be purchased at Home Depot. I purchased the Brite Aluminum Insect Screen (36" x 84") for about $7 (see **Figure 3**).

Using scissors, cut the screen into two pieces the size of 17.5" x 26" inches, so that you can install both screens together into one of the pillowcases. After the two pieces are cut — using a metal yardstick — fold over the outer four edges of each screen about 1/4" and then flatten them using the yardstick.

You must make sure that each of the cut edges of both screens are perfectly flat because if any of the screen's cut edges penetrate the insulating cloth and touches the other screen, it will short out the circuit and the alarm will sound prematurely (see **Figure 4**).

Take about 5' of copper wire and remove about 6" of the insulating plastic on one side. Weave about 3" in and out along the corner edge of the screen, then fold it back on top of itself and weave the remaining 3" right next to the first 3" of copper wire. This is done because solder will not bond to the aluminum screen like it will to the copper

■ FIGURE 7.

■ FIGURE 8.

wire. Put a little solder flux on the area with the copper wire and melt solder onto the copper wire so that all of the exposed copper wire is now soldered together.

You should now have both sides of the copper wire with the aluminum screen sandwiched in the middle, making electrical contact. Clean off the solder flux with Acetone™. Once the Acetone dries, mix some five minute epoxy and put a little over your soldered copper wire. This will help to insure that electrical contact is maintained with the aluminum screen at all times (refer to **Figure 5**). I connected up and soldered a mating jack onto the ends of the copper wire which will then plug into the alarm unit.

Next take some masking tape and attach the four outer edges of each screen. This will help to ensure that the screen's cut edges do not penetrate through the insulating cloth and touch the other screen prematurely. Masking tape worked the best out of all the varieties I tried. Electrical tape was by far the worst (see **Figure 6**).

■ FIGURE 9.

The two pillowcases that I used were 20" x 30". I purchased one pack which had a set of two for about $5 from Walmart. Sew screen 1 on the top to screen 2 on the bottom with the pillowcase in the middle. I did mine by hand, but if you have access to a sewing machine, you can do a much better job than I did.

Be careful sewing around the corners where the masking tape is the thickest and around the areas where the epoxy is. This area is too hard to sew through.

Note: Screen 2 needs to go on the bottom side of the pillowcase and not in the middle of it. I tried this on the first prototype and there was not enough insulation between the two screens which, in turn, caused the alarm to sound prematurely (refer to **Figure 7**).

When finished sewing, your screens should look like **Figure 8**, with one screen on top and the other on the bottom of the pillowcase. I made the screens a little smaller than the size of the pillowcase so you

can now slip both screens into the second pillowcase (**Figure 9**).

Ready for Testing

This is a very simple circuit but it is in a tight place. Please check and re-check all your connections with the **schematic**. If you're sure that all is good, plug the unit in and test the alarm by pushing in on the test switch #2. The piezoelectric speaker should sound and remain on even when you take your hand off of the test switch #2.

Now, take the aluminum screen sensor and measure the resistance between the two screens. When dry, they should measure no less than several megohms of resistance — if you get a reading at all. Press your hand down on the screens and the resistance should remain the same. If it drops to a low resistance, then you have a single wire (or more) from one of the screens penetrating onto the next screen. This must be located and repaired. Otherwise, you will be getting all types of premature false alarms. If everything is fine, then take a wet wash cloth and place it on top of the screen sensor. Place your hand on top of the wet wash cloth and push down. The resistance should drop to a few hundred ohms, depending on the wet area covered. Allow the screen sensor to totally dry for a day or two before testing the entire system.

When you're ready to test the entire system, plug the dry screen sensor into the alarm unit. Set pot R3 about mid way. Turn the unit on and then place a wet wash cloth onto the screen sensor and press down on it. Remember, as the screen sensor becomes wet, the resistance drops. Voltage is traveling from the top screen through the pillowcase onto the bottom screen. The voltage then travels into resistor R6 and then into pot R3. The voltage must charge up capacitor C1. This may take up to a minute or two before the alarm sounds.

Once the alarm sounds, you'll know that the system is working and ready to be used. Let the sensor screens dry, and then place them under the bed sheet where whoever is using the unit will be sleeping. You can use large safety pins in the four corners of the sensor screens to secure the sensor onto the bed mattress. If you do, you may want to take some pliers and crimp the head of the safety pins so they don't slip out of the head when you are moving around in the bed.

Stay dry, my friends. **NV**

Getting Down to Earth

Reading this article could help save you from serious injury or even death.

In general, the projects you see in electronics magazines are powered by batteries, wall mount adapters, or from a computer USB port. At some point, you might work on equipment that is powered directly from the AC power line; things like antique radios and amateur radio transmitters. The purpose of this article is to convey some little known or often overlooked information about the AC power line so that you can be aware of the possible danger as you work.

Unknown Danger

My first job out of college was as a development engineer for a company that made large industrial motor drives up to 1,000 HP. I worked in a laboratory that had many prototypes and breadboards out in the open on top of work benches. Most of the circuits were powered from 240V or 480V AC power lines. One day I was working on a drive that was acting erratic, so I decided to look at the gate drive waveform with an oscilloscope. As I connected the clip lead of the scope probe to the gate drive circuit board, I was temporarily blinded by a flash of light. When I could see again, I found that the palm of my hand was black and slightly burned. The probe clip lead was gone as were most of the copper foil traces on the printed circuit board. I was very lucky in that my injuries were minor and short lasting.

As I went about trying to understand what had happened, I learned about a subject that was not taught in engineering school but is well known to every real electrician: GROUNDING. In many parts of the world, what we in the US call ground or grounding is called earth or earthing. Grounding systems and practices are intended to make things powered by AC outlets safer. When someone is working on the guts of an AC powered device, the grounding system must be fully understood in order to work safely.

My accident occurred because the metal enclosure of the oscilloscope and the probe clip lead were connected through the ground pin on the AC line plug to the building ground system. One side of the 480V power line was also connected to ground through the neutral conductor, so basically, I had shorted out a 480V circuit with the oscilloscope clip lead. I was very lucky that the foil traces on the circuit board acted as a fuse and quickly interrupted the current before really serious injury occurred. Modern oscilloscopes are in plastic cases, but if you measure the resistance between the probe clip lead and the ground pin on the AC power cord you will find that they are connected.

Later in this article, we will explain how to use an oscilloscope safely, but first read on so that you totally understand grounding as it applies to the AC power line in your house.

Technical Language

Technical language is meant to be very precise when used properly. Unfortunately, it is often misused by both non-technical and technical people alike. We tend to throw around terms like heat and temperature or force and pressure as though they are the same. We often say "Turn up the heat" or "Apply pressure to the pushbutton," etc.

Does your car have a negative ground system? If your

Post comments on this article and find any associated files and/or downloads at www.nutsvolts.com/magazine/article/June2016_Earth-Grounding-Techniques.

car is like mine, it is mounted on four doughnut shaped insulators and it is not connected to ground at all (except when it is connected to a battery charger). For safety's sake, we should be more precise when talking about ground, chassis connection, and circuit common. We often treat them as though they are the same thing when they actually are quite different.

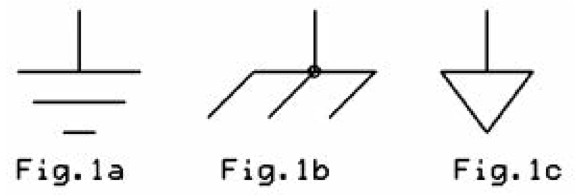


FIGURE 1. Symbol a is for earth ground. Symbol b is for a connection to a chassis or enclosure. Symbol c is for a circuit return or common.

Symbols

We use a lot of symbols in electronics. We all know that we don't use the symbol for a resistor to indicate a capacitor, or the symbol for an inductor to indicate a transistor, but for some reason we interchange the symbols shown in **Figure 1** all the time. Let's discuss each symbol individually.

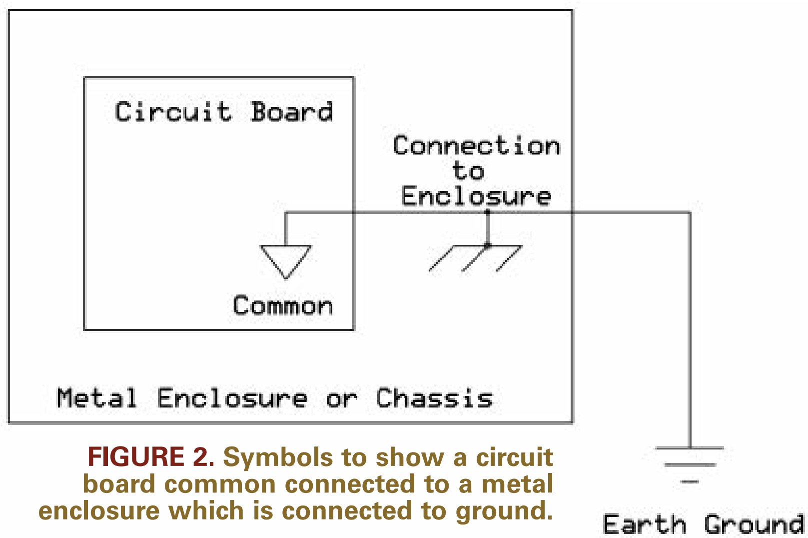


FIGURE 2. Symbols to show a circuit board common connected to a metal enclosure which is connected to ground.

Ground

Figure 1a is the symbol for an electrical connection to ground (mother earth). The component that is indicated by this symbol is shown in **Photo 1**. In the US, this symbol is for an eight foot long copper covered steel rod that has been driven into the ground/earth. You can find this electrical component below or near the electric meter on the outside of your house. Sometimes there may be more than one ground rod, depending on the age of your house and the local electrical code. A copper wire connected to the ground rod runs to the electrical distribution box inside the house. It may first go to the base of the electric meter and then to the distribution box.

The ground symbol can also be correctly used on a schematic diagram to indicate the earth ground of an antenna system, but here again it is indicating a rod driven into the earth. You may find that your cable company or your land line telephone company has also connected their ground system to this rod.

The latest version of the national electrical code requires that the wire that goes between the ground rod and the meter base must pass through a device called an "intersystem bonding terminal." This is a short bus bar with several terminals that the other services are supposed to use to ground their systems.

The symbol of **Figure 1a** is used on equipment to indicate terminals that are to be electrically connected to an earth ground, as well. I have seen this symbol used on representative schematics of integrated circuit chips and on schematic diagrams of battery powered circuits that have nothing to do with earth ground. I have even seen this symbol used on diagrams of satellite and balloon born circuitry. (I guess I'm overlooking the ground wire that runs between satellites and the earth.) If you page through technical magazines that have to do with electronics, you will find this symbol used incorrectly to indicate circuit returns (or common) on all kinds of ungrounded circuits. It is obvious that many folks working in the electronics field have no idea what this symbol denotes. The same people

PHOTO 1. Actual earth ground. This component is represented by the symbol shown in Figure 1a.

that create a datasheet for an integrated circuit chip with this symbol on the chip diagram will sometimes actually label a pin as "GND." No wonder this subject is so confusing to many individuals working with electronics.

Chassis Connection

Figure 1b is the symbol that indicates a connection to a chassis or electrical enclosure. This symbol is sometimes referred to as the "crow's foot" (at least by some of us old guys). This symbol is seldom seen in electronics magazines, but it is the symbol that should be used to indicate connections to the chassis of your car, the frame of an aircraft, the skin of a satellite, or the metal box that your latest project is in. The actual connection is usually done with a ring terminal (lug) and a screw.

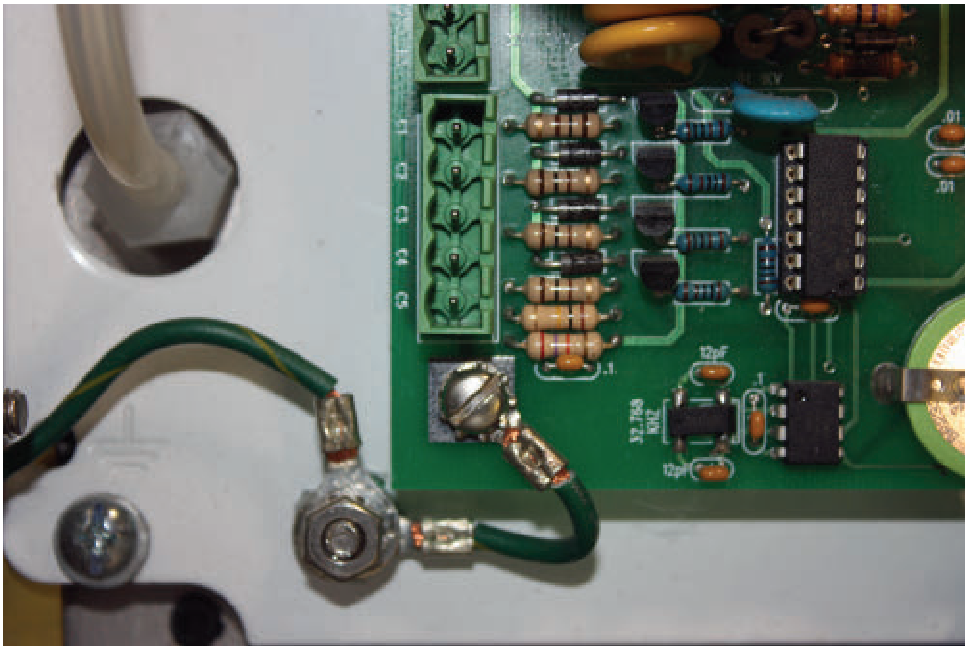


PHOTO 2. Typical connection of a circuit common (Figure 1c) to an enclosure (Figure 1b).

Photo 2 shows a typical connection between a circuit board and its electrical enclosure. Sometimes this symbol has nothing to do with grounding. A circuit board may be mounted in a metal enclosure for shielding reasons. If the circuit is battery powered, there may be no reason to "ground" the enclosure.

Common (Circuit Return)

Figure 1c is the symbol used to designate the power return or common of an electronic circuit. This is the symbol that should be used in 99.99% of the cases where you see the ground symbol (**Figure 1a**) used. This symbol is sometimes modified with a letter or an asterisk inside the triangle to indicate different types of returns that must be kept separate, such as signal common vs. power common, etc. When power common and signal common are mixed together, interference can be encountered through what is commonly referred to as a "ground" loop. This term does not apply when the circuit has nothing to do with a path to ground. The proper term would be a return loop. The explanation of return loops is beyond the scope of this article, but I wanted to show another incorrect usage of the word "ground."

Horse Power

In many modern housing developments, the utilities are underground. In rural areas — where the power lines are on utility poles — you can see cylinder shaped transformers mounted on some of these poles. **Photo 3**

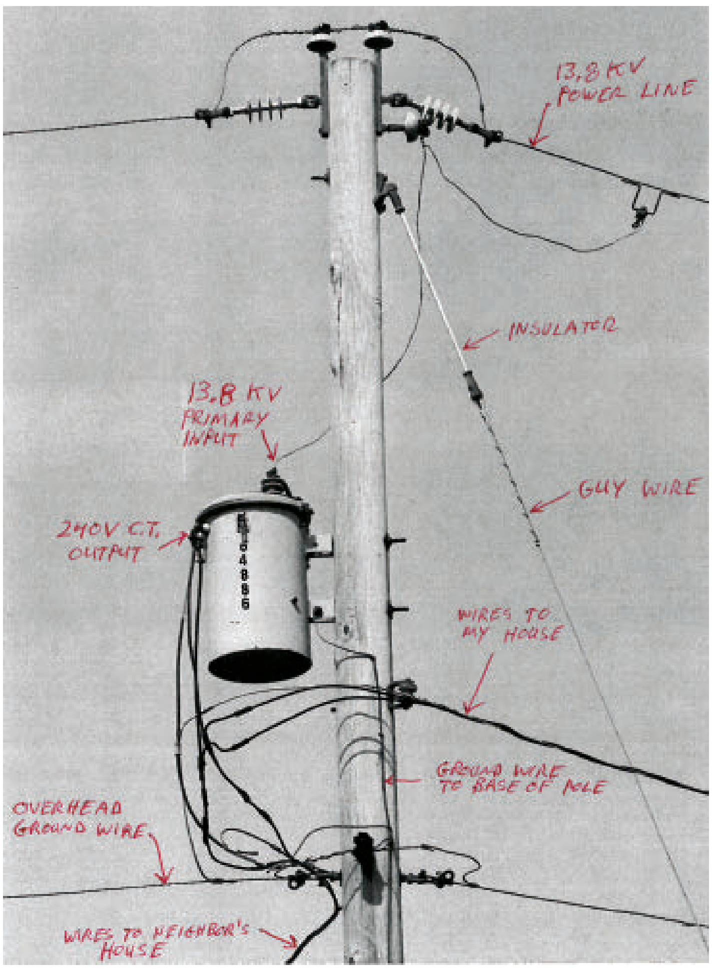


PHOTO 3. Typical power transformer installation.

shows a typical pole mounted transformer installation. These transformers are sometimes called pole pigs by people familiar with them. The purpose of these transformers is to reduce the distribution voltage down to the center tapped 240V that is used to power a house, and at the same time to isolate the secondary 240V from the primary voltage. The primary distribution voltage can range from 2,300V to as high as 39,000V. **Figure 3** is a diagram of the transformer circuit.

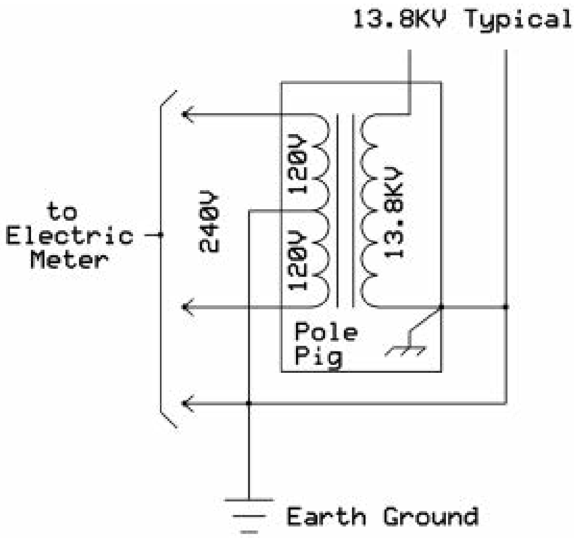


FIGURE 3. Wiring of the transformer that is feeding your house.

If the houses are far apart, there will be one transformer for each house. When houses are close together, a single transformer may feed several houses. The utility company grounds the center tap of the secondary of the transformer through an overhead ground line and a bare wire that runs down the utility pole and into the ground. You can see in **Photo 3** that the grounded neutral conductor from the center tap is connected to both the overhead ground wire and the wire that runs down the pole. Also note that the case of the transformer is connected to these wires. The bare neutral conductor is often used as a support for the line conductors that run to a house.

Most electrocutions occur when a grounded person comes in contact with a live wire. If the electrical system in your house was not grounded, touching a live wire while you were grounded would not cause any harm. This might lead you to believe that things would be much safer if the electrical system in your house was not grounded, but this is not the case.

The transformer that supplies power to your house isolates the household wiring from the high distribution voltage that is connected to the primary of the transformer; however, if the transformer insulation should become leaky or totally break down, your house wiring could float at up to 39,000V above ground. This makes 120V seem rather safe in comparison. This is the main reason your house wiring is grounded in the first place. Grounding systems can also help to channel lightning strikes to ground and can prevent the buildup of static charge from the earth's electric field or from the wind.

Figure 4 is a simplified diagram of the wiring of a house showing only one of many outlets. Electrical wiring practices have changed over the years, so it depends on the age of your house as to how the wiring in the main electrical enclosure was done. **Photo 4** shows the inside of a fairly modern main electrical enclosure. The black (hot or live) wires are fed from circuit breakers. Notice that some of the breakers have an extension with a white label. These extended breakers are ground fault circuit interrupters (GFCI). These breakers are used for circuits that feed potentially wet areas such as bathrooms and kitchens.

The white (neutral) wires from circuits that are fed from regular breakers are connected to the neutral bus bar. The white wire from a circuit that is fed from a GFCI is connected directly to the neutral terminal on the GFCI. This neutral circuit goes through the GFCI detection transformer and then to the neutral bus bar. The bare (ground) wires should run to a different bus bar than the neutral wires, but in many houses the white (neutral) wires and bare (ground) wires are all connected to the same bus. Inside the main electrical enclosure (or in the meter

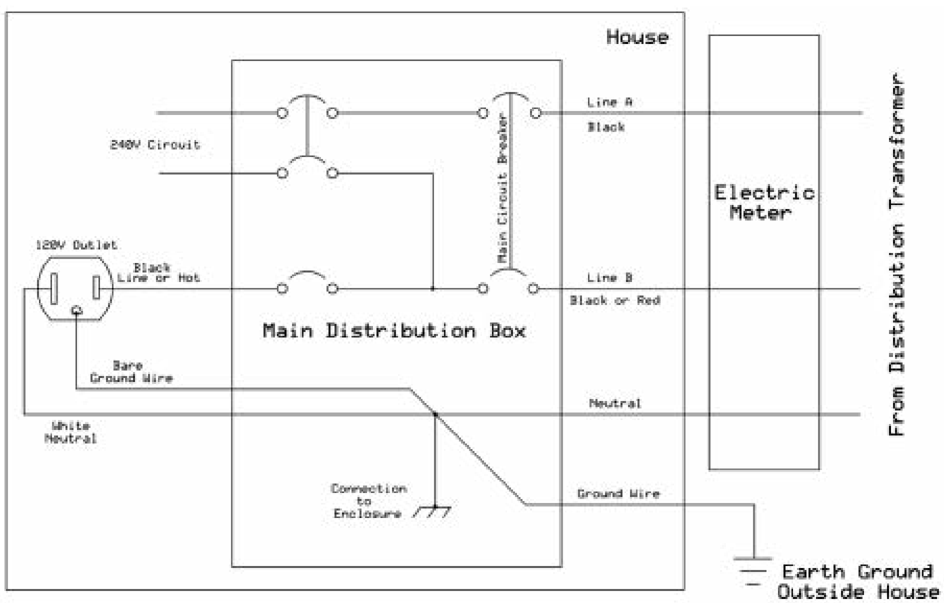


FIGURE 4. Simplified schematic of a mains power distribution box.

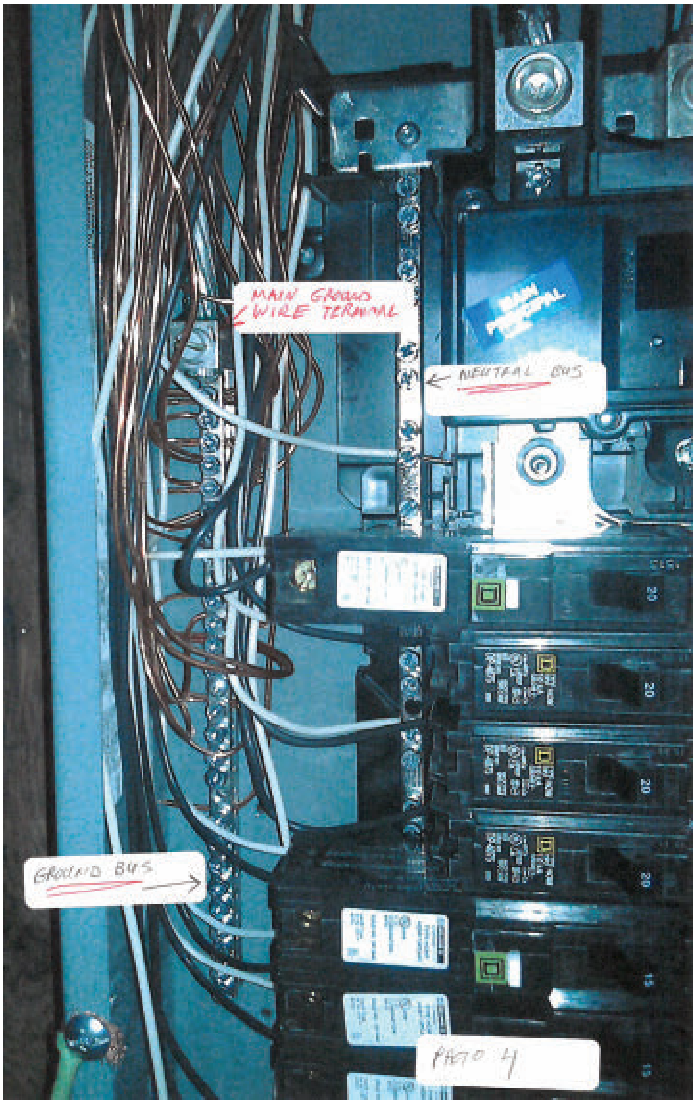


PHOTO 4. Typical wiring of the mains distribution box.

base), the neutral wires, the ground wires, and the enclosure itself are connected together and to the ground electrode conductor that runs to the ground electrode (ground rod.) If the water supply to your house was done with copper pipe, you may also find a wire from the ground circuit connected to the copper water pipe near where it enters the house. There are many variations due to age and location of the house, but the main point is that the neutral wires are connected to the ground system only where the power enters the house.

Now, you might ask, since the neutral wires and the ground wires are connected together, why have both? The reason is that the neutral wires are current carrying wires and the ground wires are a personnel protection system. The metal skin of your washer, dryer, refrigerator, microwave, etc., is connected to the ground system. This is to protect you in case a failure inside the appliance allows a hot wire to come in contact with the metal parts of that appliance. Such a failure will cause a breaker to trip and disconnect the appliance from the AC line.

Sometimes insulation of components inside an appliance will become leaky and there could be a high impedance path to the metal parts. In this case, the current that flows could be less than what is needed to trip the breaker. If the appliance was not grounded, the metal parts could be at 120V above ground. If you touched the appliance while grounded, you could be shocked or electrocuted.

Ground fault circuit interrupters come in two types. The GFCI can be part of a circuit breaker in an electrical enclosure or it can be built into an electrical outlet. Sometimes the "C" is left out and the slang term GFI is used. The GFCI can detect if the hot wire is carrying more current than the neutral wire, as would be the case if a grounded person came in contact with a hot wire. The GFCI will quickly disconnect the outlet from the line if this should happen. The operation of a GFCI would not be possible if the neutral wires and the ground wires were not kept separate.

Working on Hot Stuff

The following may not cover all situations and eventualities. Consult other sources for complete information on this subject.

Sometimes it is necessary to measure voltages or look at waveforms with the power on. Before you probe around inside a piece of equipment with the power on, be sure to take the utmost precaution.

Ground fault circuit interrupters are required for circuits in wet areas like kitchens and bathrooms and outdoor outlets. If there is any possibility that you might work on AC line powered equipment, it would be a good idea to change the power outlets in your work area to GFCIs. If your work area has a concrete floor, you should invest in a large rubber mat to insulate you from the floor. (A mat is actually a good idea on any type of floor.) Most rubber footwear is a fairly good insulator but only if it is dry. Never work on AC line powered equipment unless your shoes are dry!

Be aware of any grounded metal objects in your work area. Is your workbench made of metal or does it have metal legs? Are you sitting on a metal stool or chair? Make sure you are not in contact with any grounded metal objects as you work. Do you have an AC line powered computer on your bench? Be aware that its metal case is grounded as are the metal parts of those USB connectors, as well as the shield on the headphone connector.

Wear eye protection. You could accidently cause a short with a meter probe that might send sparks towards your face. Also be aware that trimming the end of a solid wire (like a component lead) with diagonal cutters will

send the small pieces that were cut off flying away at a fairly high velocity. Take care that this does not hit you in the face or end up inside the equipment that you are working on.

Be aware that some equipment may have internal voltages far greater than the 120V AC power line. Tube type radios may have voltages several times higher than 120V. Televisions and computer monitors with picture tubes, amateur radio transmitters, microwave ovens, and copy machines can have several thousand volts on internal components, and capacitors in these devices can hold these high voltages for long periods of time after power is turned off. Never work on this type of equipment unless you know exactly what you are doing.

Work with only one hand. Keep one hand behind your back and make sure you are not touching a grounded object when working on live equipment.

Test Equipment

Any piece of test equipment that is powered from the AC line is most likely grounded. (Don't repeat the accident described at the beginning of this article!) You can safely look at waveforms with a two-channel scope by putting it in the differential mode (A minus B) and using the two probes without the ground clip leads. Just remember the oscilloscope is grounded. Do not come in contact with its case when working on a live circuit.

The knobs on some oscilloscopes are attached with set screws. These set screws must also be considered to be grounded objects. Luckily, there are apps and ancillary hardware that can turn battery powered devices like cell phones, tablets, and portable computers into test equipment. These devices can make working on AC line powered equipment safer, but only if you know what you are doing.

Summary

I hope this article has made you more aware of the purpose of grounding and at the same time the dangers of grounded objects when working on live circuits.

We can help to make things safer for everyone by using the proper terms when talking about ground, chassis connections, and circuit common. We may never be able to end the misuse of these terms, but we can at least use the correct symbol for each of them from now on.

Work safely, my friends. **NV**

... I'm an analog man

RELIABLE IoT DATA ON YOUR DESK

By John Rucker

There is just something about a good ol' analog panel meter that conveys trust in the value being presented. You can't get much more reliable than a 50 uA panel meter connected to a DC shunt. If the needle was off zero, you just knew it was working. If the value wasn't what you expected, a couple of taps on the faceplate would reassure you the value was true.

I will never forget the first time I worked with a panel meter. I was amazed at the simplicity and how customizing its placard made any project look professional. We could sure use some of this trust in the presentation of today's IoT (Internet of Things) cloud based data.

Just because the days of a simple self-powered panel meter are long gone doesn't mean we have to accept a displayed value as more or less accurate! So, I set out to connect an analog panel meter to the IoT and bring back the trust in the value displayed.

One of the advantages of having a value displayed on a physical analog panel meter is it only takes looking at it to read it. No finding your phone, opening an app, selecting the device, and loading its value. Simply glance across your desk and look. I wanted my IoT sourced panel meter to be true to that form. One glance is all it should take to know if the value being displayed is correct and within acceptable parameters (not too high or low). That turned out to be the real challenge to this project.

How can I convey trust that the value being displayed on the panel meter is valid with a quick glance? If it's not valid, I need to give the user a simple way to "tap on the faceplate" and request valid data.

To do this, I added a marquee to the top of the panel meter's box. The marquee is lit by a row of WS2812 color addressable LEDs. I can change the color and brightness of the marquee to

in a digital world ...

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/June2016_Analog-Panel-Meter-IoT-Data.**

represent different states of the data being displayed on the panel meter. This way, in one glance, the user can read the value of the panel meter and tell if the data is valid based on the marquee's color. What if you're not looking at the marquee and your data is not within acceptable parameters? Well, that one is simple. I added a little piezo buzzer for that with settable values for the audible alarm. To mute the alarm and request data updates ("tap on the meter face"), I put a little red button on the back of the box.

IoT SOURCED PANEL METER ARCHITECTURE

My panel meter is going to source its data from the IoT cloud (not a locally connected DC shunt), so I need to add a microcontroller and a radio. For that, I'm going to use the same hardware and software library as in my January 2016 *Nuts & Volts* article (SmartThings® and the Device Maker) that is based on the Parallax Propeller and Digi's xBee ZigBee radio.

I want the panel meter to be generic in nature so it can be used to display values from just about any source: wind-speed, temperature, LUX, current, river level, and even the state of a door (open or closed). After all, since you can customize the meter's placard, you can make the little needle measure anything.

Zigbee Clusters and Microcontroller Firmware

ZigBee clusters provide an application layer for device communications. They define how your device should communicate by detailing the radio's data packet all the way down to the byte level. This may seem tedious, but by following the cluster's specifications two completely independent vendors can make devices that communicate with each other. This allows your project to control or be controlled by other off-the-shelf ZigBee devices. This standardization is one of the reasons ZigBee is so popular in the home automation market.

ZigBee clusters sit on top of the ZigBee networking protocol. You **can** have one without the other; for example, the Thread Group© is going to support ZigBee clusters on top of their wireless protocol. So — from an application standpoint — if your code formats its data according to a standard ZigBee cluster definition, there is a good chance it won't be a huge undertaking to port it over to a Thread network.

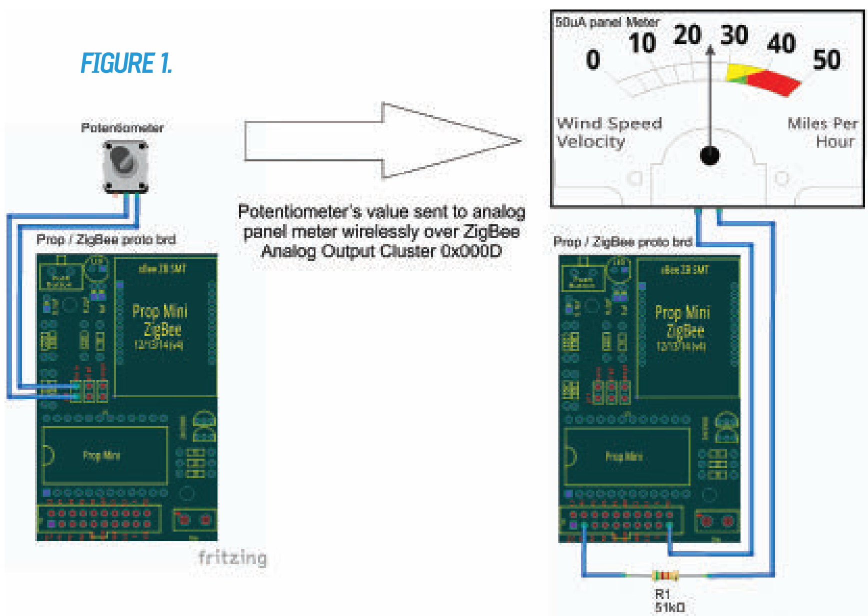
All public ZigBee clusters are defined by the ZigBee® Alliance and you can freely download a ZigBee Cluster Library (ZCL) specification from the **ZigBee.org** website. The ZigBee Analog Output cluster number 0x000D just happens to be a perfect fit for the analog panel meter! It was designed for almost this exact purpose. A quote from the ZigBee Cluster library page 168 section 3.14.2.1 states, *"The Analog Output (Basic) cluster provides an interface for setting the value of an analog output (typically to the environment) and accessing various characteristics of that value."*

A cluster's definition will consist of a list of attributes and commands. Attributes are place holders for data and the type of data they contain. For example, our Analog Output Cluster (cluster number 0x000D) has an attribute called "PresentValue" (attribute number 0x0055) with the data type of single precision. So, if we write our code to listen for data at that attribute number, any device on the ZigBee network that adheres to the Analog Output cluster can control our meter's needle.

PACKET DETAIL

Say we have two wireless devices on our ZigBee network. The first device is a simple potentiometer connected to a microcontroller and a ZigBee radio (left side of **Figure 1**). The second device is a microcontroller connected to a ZigBee radio and an analog panel meter (right side of **Figure 1**). As the user turns the potentiometer, we want to send that analog value to the panel meter. Let's assume the potentiometer is set halfway

FIGURE 1.



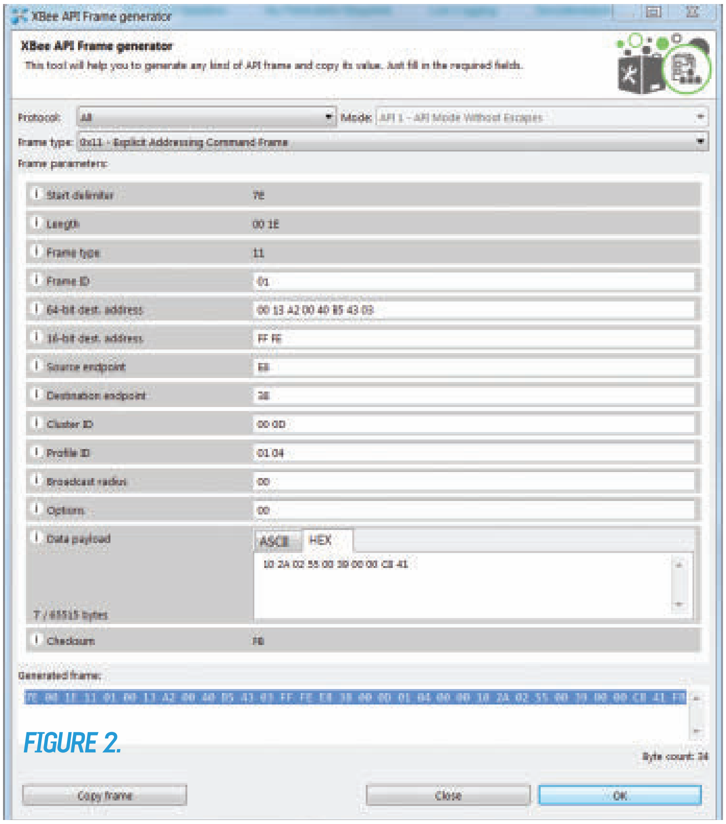


FIGURE 2.

between its maximum and minimum value, and that represents the value of 25. How do we send the value of 25 to our panel meter? Let's break it down.

We will have to communicate with the ZigBee radio in API mode to send the ZigBee cluster data. In this case, we are using Digi's xBee ZB radios and will be using the format of an xBee Explicit Addressing Command frame (frame type 0x11) on page 146 of the *ZigBee RF Modules User Guide*.

The microcontroller on the left will send the following hex byte array to the xBee, to send the value 25 to the analog panel meter on the right:

```
7E 00 1E 11 01 00 13 A2 00 40 B5 43 03 FF FE
E8 38 00 0D 01 04 00 00 10 2A 02 55 00 39 00
00 C8 41 FB
```

Let's take a look at the above packet using Digi's XCTU xBee API Frame generator shown in **Figure 2**.

As you can see, most of the byte array consists of addressing fields that don't change much from packet to packet. Take a look at the cluster ID and note that it is set to 0x000D (our analog output cluster number). Also note the destination endpoint number of 0x38. It gives us some very interesting features I will get back to later. If you take away the network and the application layer addressing parameters from the byte array, you're left with just 10 bytes that make up the actual data payload (in [square brackets]):

```
7E 00 1E 11 01 00 13 A2 00 40 B5 43 03 FF FE
E8 38 00 0D 01 04 00 00 [10 2A 02 55 00 39 00
00 C8 41] FB
```

These 10 bytes are defined in the cluster definition for the analog cluster 0x000D and they parse out as in **Table 1**. The analog meter's microcontroller will need to parse this data to pull out the 25.0 floating point value. To do this, I first parse out just the 10 byte data payload (the data payload mentioned above) from the xBee's data packet into a byte array, then look at byte 3 to see what the command is for this packet. In this case, the command is 0x02 "write attribute" (see Table 2.9 on page 16 of the ZCL for a full list of command numbers). The two most common command numbers are 0x02 for the write attribute and 0x00 for the read attribute. So, now we know this is a write command and the next two bytes will be the attribute number to write.

In this case, it is 0x0055 and we know that attribute is the *present value* from our 0x000D cluster definition. I double-check that the data type value is correct and, if it is, I then convert bytes 7-10 to the correct endianness 0x41C80000 and store it in a floating point variable that equals 25.0. That variable is then passed to the method that drives the analog meter and sets the needle to the correct position.

Table 1.

Byte #	1	2	3	4	5	6	7	8	9	10
Data Payload	10	2A	02	55	00	39	00	00	C8	41

10 2A 02	Frame control header	See section 2.3.1.1 on page 14 of ZCL. 0x10 = Cluster wide command and disable default response. 0x2A = Transaction sequence number. 0x02 = Write attribute command; see section 2.4.3 on page 21 of ZCL.
55 00	Attribute to write in little-endian format	0x0055 = Present value (see Table 3.68 on page 168 of ZCL).
39	Data type	0x39 = Single precision floating point value will follow (see Table 2.16 on page 56 of ZCL).
00 00 C8 41	The value 25 as four-byte floating point in little-endian format	The hex number 0x0000C841 is in little-endian format; reverse the byte order to get 0x41C80000. 0x41C80000 is the IEEE 754 encoded value of 25.0 as a decimal.

ZigBee Destination Endpoints

Now, let's get back to that destination

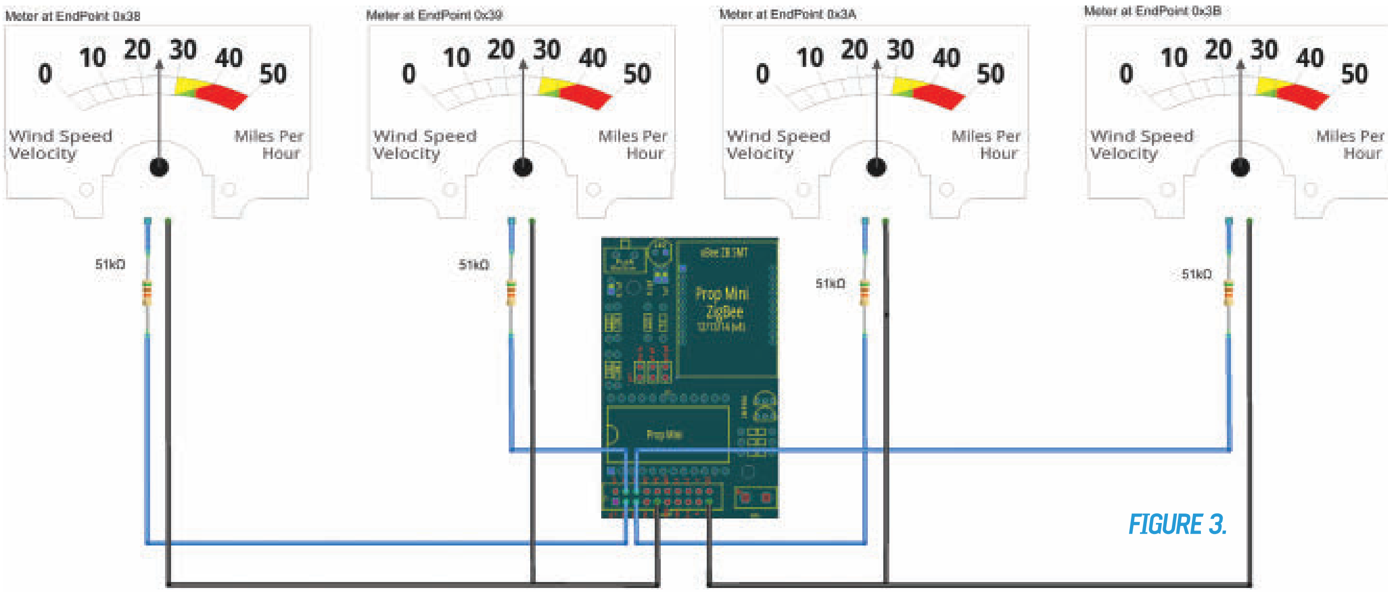


FIGURE 3.

endpoint number 0x38. A ZigBee endpoint number along with the profile and cluster number give you application layer addressing. This allows you to direct a command to a specific component of your device. If we add three more analog panel meters to our microcontroller (giving us a total of four), we can assign each one a unique endpoint as shown in **Figure 3**. That is exactly how we send data to our four panel analog meter.

Here is what the packets would like if we wanted to send the value 25 to all four meters at endpoints 0x38, 0x39, 0x3A, and 0x3B:

```
7E 00 1E 11 01 00 13 A2 00 40 B5 43 03 FF FE E8
[38] 00 0D 01 04 00 00 10 2A 02 55 00 39 00 00
C8 41 FB
7E 00 1E 11 01 00 13 A2 00 40 B5 43 03 FF FE E8
[39] 00 0D 01 04 00 00 10 2A 02 55 00 39 00 00
C8 41 FA
7E 00 1E 11 01 00 13 A2 00 40 B5 43 03 FF FE E8
[3A] 00 0D 01 04 00 00 10 2A 02 55 00 39 00 00
C8 41 F9
7E 00 1E 11 01 00 13 A2 00 40 B5 43 03 FF FE E8
[3B] 00 0D 01 04 00 00 10 2A 02 55 00 39 00 00
C8 41 F8
```

This gives each panel meter its own unique address, allowing any ZigBee device to send an analog value to a specific panel meter all on one microcontroller. So, if we had a third ZigBee device that was hooked up to a thermistor, it could send its temperature value to panel meter 2 by sending the packet to endpoint 0x39. You can have over 250 unique endpoints, so it is possible to hook up 250 analog panel meters to our microcontroller and address them all with unique endpoints. Of course, that's not feasible with the microcontroller we are using here, but you can see the expansion potential by making our application endpoint addressable.

Another relationship can be drawn between an endpoint and a method within your firmware. Basically, endpoints give you unique addresses for methods within your microcontroller. Once I got my head around that it was a real ah-ha moment for me.

Vendor-Specific Attributes can be Customized to Your Device

As we discussed earlier, each cluster definition has a defined list of attributes that allow you to send and receive data (i.e., Attribute 0x0055 = *present value*). The cluster's specification also sets aside a range of attribute numbers for "vendor-specific attributes." Vendor-specific attributes allow you to build a custom solution on top of a standard cluster; it is up to you to document them or not. Often, vendors use these attributes and don't document their usage. That is fine, as long as they are not using standard attribute numbers. You can do whatever you like with reserved attribute numbers. You can find these reserved numbers for the analog out cluster in Table 3.68 on page 169 of the ZCL. Cluster 0x000D has attributes from 0x0400 to 0xFFFF reserved for vendor-specific functions.

Since the goal of the four panel meter is to make it generic so it can display a floating point value that represents any type of data, we must be able to tune each meter. Let's say meters 1 and 2 (end points 0x38, 0x39) are going to display wind speed, and their scale will range from 0 to 50 MPH. Meters 3 and 4 (end points 0x3A, 0x3B) are going to display temperature, and their scale will range from -15° to 120° Fahrenheit.

In order for our microcontroller to position the needle correctly on each meter, it needs to know what the meter's minimum and maximum values are to calculate the position. These minimum and maximum values would fit perfectly in a custom attribute. **Table 2** lists the attributes used by the four panel meter. Attributes 0x0406

Table 2					
Four Panel Meter Attribute Definition					
Attribute #	Description	Data Type	Data Description P55 of ZCL	Read/Write	Default
0x0055	Display current value	0x39	Floating point single precision	R/W	0
Custom Attributes					
0x0400	Set marquee default color	0x23	Unsigned 32-bit integer	R/W	0x0E0600
0x0401	Set marquee default level	0x23	Unsigned 32-bit integer	R/W	200
0x0406	Meter min scale value	0x2B	Signed 32-bit integer	R/W	0
0x0407	Meter max scale value	0x2B	Signed 32-bit integer	R/W	50
0x0408	Meter max FRQ (set at factory)	0x23	Unsigned 32-bit integer	R/W	
0x0500	Data time out in seconds	0x23	Unsigned 32-bit integer	R/W	
0x0501	Low alarm value	0x2B	Signed 32-bit integer	R/W	-1
0x0502	Warning value (yellow light)	0x2B	Signed 32-bit integer	R/W	30
0x0503	High value (red light)	0x2B	Signed 32-bit integer	R/W	35
0x0504	Major alarm (flashing red light)	0x2B	Signed 32-bit integer	R/W	45

and 0x0407 hold the min/max value for each meter. This allows us to programmatically change the configuration of our four panel meter. You can also see I used attributes 0x0500 to 0x0504 to set alarm values for each panel meter.

By making the alarm values settable via a custom attribute, it is now possible for users to easily change the value that will cause an alarm. For example, let's say panel meter 4 is being used to monitor the water temperature of your pet's outside heated water bowl. You could set the low alarm to go off when the value is below 32°F letting you know there is a danger of the water freezing over.

Come springtime, you move that probe to monitor the heat from a heat lamp pointed at your daughter's baby chicks out in the garage. You now want to know if the low value ever drops below 80°F as that will be too cold for baby chicks. Since you can change the low alarm value per panel meter by updating custom attribute 0x0501, you can easily handle both applications all by sending one packet to update a custom attribute on that meter's end point.

Tip: When I create a new device, I have found it really helps to define your custom attribute list early in the process. Not only do the attribute definitions make a good reference point for your program's variables, I often take this table and cut and paste it into my device's documentation.

Zigbee Network Layer Communications

Earlier I pointed out ZigBee clusters are the application layer for a ZigBee device. To get our IoT panel meter properly connected to a network, we need to focus a little on the network layer. The ZigBee Device Objects (ZDO) library is a collection of clusters for sending and receiving ZigBee network layer packets. Sending a ZDO cluster command is very similar to sending a ZCL command, but with two small changes.

To send a ZDO packet, you have to set your profile ID to 0x0000 and your destination endpoint to 0x00. This tells your radio to treat the data packet as a ZDO command. ZDO commands are extremely powerful. They allow you to bind devices together, create groups, check signal strength, read routing tables, and query devices for their capability. To be properly identified on a ZigBee

Table 3				
Four Panel Meter	Cluster		Cluster	Hub (ZigBee Coordinator)
Device Announce (broadcast)	0x0013	→		
		←	0x0005	Report Active End Points request
Active End Point Response	0x8005	→		
		←	0x0004	Simple Description Request for EP 0x38
Simple Description RPT for EP 0x38	0x8004	→		
		←	0x0004	Simple Description Request for EP 0x39
Simple Description RPT for EP 0x39	0x8004	→		
		←	0x0004	Simple Description Request for EP 0x3A
Simple Description RPT for EP 0x3A	0x8004	→		
		←	0x0004	Simple Description Request for EP 0x3B
Simple Description RPT for EP 0x3B	0x8004	→		

For a step-by-step build guide, please see the instructions included in the IoT Panel Meter kit that's available in the *Nuts & Volts* webstore.

You can see a video of the IoT panel meter at **https://youtu.be/Ubivg4C9L80**.

network, our IoT panel meter needs to support at least three ZDO commands:

- **Device Announce** (ZDO cluster 0x0013; see page 111 of ZDO spec). When a ZigBee device joins a ZigBee network, it needs to announce itself. This cluster is broadcast to all non-sleeping devices on the network letting them know it's available to receive data. Often, a device announce will trigger an identification process from the ZigBee Coordinator (hub).
- **Active End Point Response** (ZDO cluster 0x8005; see page 163 of ZDO spec). When the hub receives a device announce packet, it will send a ZDO cluster 0x0005 report active end point request. The IoT panel meter will listen for this request and respond with a list of its active end points by sending a ZDO Active End Point Response packet 0x8005.
- **Simple Descriptor Response** (ZDO cluster 0x8004; see page 161 of ZDO spec). Once a list of endpoints has been received by the hub, it will then ask for details on each end point by sending a ZDO cluster 0x0004 report simple descriptor. The IoT panel meter will listen for this request and respond with the details for each end point in a ZDO cluster 0x8004 formatted packet. This packet will contain a list of clusters supported by that end point.

Table 3 shows what the ZDO packet exchange will look like when the IoT panel meter connects to the ZigBee network.

After this exchange, the hub will know what this device is and how to communicate with it. This occurs every time the device is powered on.

Device Announce Packet Detail (ZDO cluster 0x0013)

Figure 4 shows the details of the Device Announce 0x0013 packet the IoT panel meter sends when it first connects to the ZigBee network:

```
7E 00 20 11 01 00 00 00 00 00
00 FF FF FF FD 00 00 00 13 00
00 00 00 [22 79 3D 03 43 B5
40 00 A2 13 00 8C] 8C
```

This looks very similar to our previous ZCL packet, but take notice of the destination end point and Profile ID which are both set to 0, making this a ZDO command. Again, the real meat of this packet is the 12 bytes that make up the data payload in [square brackets] above. They parse out according to the ZDO cluster 0x0013 specification page 111, shown in **Table 4**.

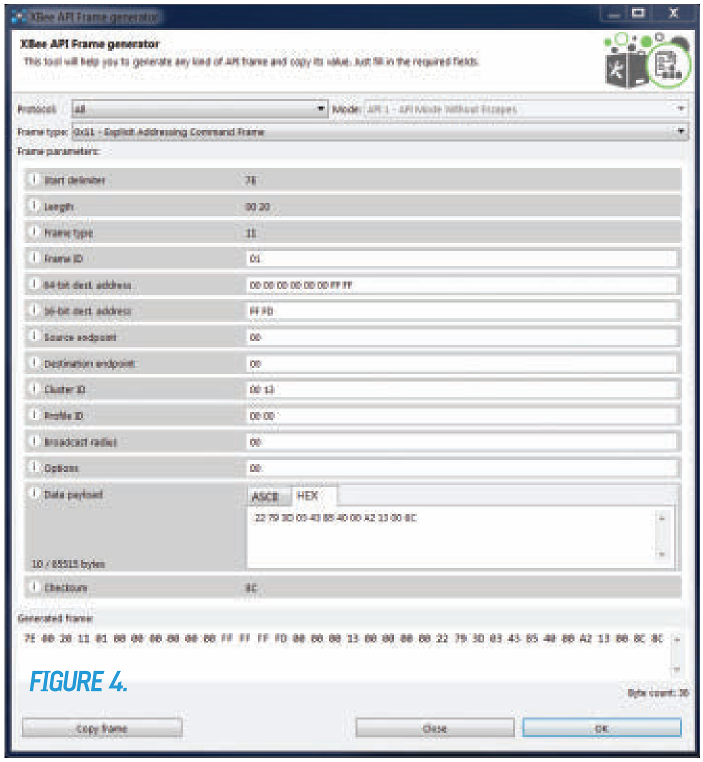


FIGURE 4.

Active End Points Response Detail (ZDO cluster 0x8005)

Figure 5 shows the active end points report sent by the four panel meter in response to the report active end points request from the hub:

<table>
<tr><th colspan="13">Table 4</th></tr>
<tr><td>Byte #</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr>
<tr><td>Data Payload</td><td>22</td><td>79</td><td>3D</td><td>03</td><td>43</td><td>B5</td><td>40</td><td>00</td><td>A2</td><td>13</td><td>00</td><td>8C</td></tr>
<tr><td>22</td><td colspan="4">Sequence number</td><td colspan="8">See section 2.4.2.8.1 on page 97 of ZDO specification.</td></tr>
<tr><td>79 3D</td><td colspan="4">Network address
Two-byte little-endian</td><td colspan="8">This is the four panel meter's two-byte network address in little-endian format. It must be read from the xBee during your firmware boot process just after it joins the network as this address is dynamic and will change.</td></tr>
<tr><td>03 43 B5 40 00 A2 13 00</td><td colspan="4">I3E Network address
Eight-byte little-endian</td><td colspan="8">This is the IEEE address (MAC address) of the four panel meter's xBee.</td></tr>
<tr><td>8C</td><td colspan="4">MAC capability flag field</td><td colspan="8">See section 2.3.2.3.6 on page 83 of ZDO specification.
0x8C = b10001100 = Mains powered device; receiver on when idle; address not self-assigned.</td></tr>
</table>

Table 5

Byte #	1	2	3	4	5	6	7	8	9
Data Payload	06	00	79	3D	04	38	39	3A	3B

06	Sequence number	Sequence number should be set to match the sequence number of the requesting packet.
00	Status	00 = Success
79 3D	Network address Two-byte little-endian	This is the four panel meter's two-byte network address in little-endian format. It must be read from the xBee during your firmware boot process just after it joins the network as this address is dynamic and will change.
04	Active end points	The number of active end points on this device.
38 39 3A 3B	End point list	List the end point numbers supported by this device.

```
7E 00 1D 11 01 00 00 00 00 00 00 00 00 FF FE 00
00 80 05 00 00 00 00 [06 00 79 3D 04 38 39 3A
3B] C5
```

The breakdown of the active end point response data packet cluster parses out according to the ZDO cluster 0x8005 specification page 163, detailed in **Table 5**.

Simple Description Report Detail (ZDO cluster 0x8004)

Figure 6 shows the packet details of the simple description report for end point 0x3B. This report is sent four times; once for each end point as requested by the hub:

```
7E 00 23 11 01 00 00 00 00 00 00 00 00
FF FE 00 00 80 04 00 00 00 00 [0A 00 79
3D 0A 3B 04 01 07 00 00 00 01 0D 00] 4D
```

The breakdown of the Simple Description report for end point 0x3B parses out according to the ZDO cluster 0x8004 specification page 161, detailed in **Table 6**.

These are the most popular ZDO packets that your device will have to process. There will be other packets to bind your device to the hub, but for the most part the xBee ZB SMT radio will handle these automatically. At this point, the four panel meter is completely connected to the ZigBee network and properly identified. It is ready to start receiving data.

Accessing the Internet of Things

Up to this point, we have been displaying data from locally attached ZigBee devices. By connecting our four panel meter to a SmartThings® hub (ZigBee coordinator), we open up access to an Internet full of data via their online development environment. SmartThings is much like other home automation solutions on the market today in that they have a ZigBee hub that sits on your home network and connects to their cloud for control. Your smartphone connects to the SmartThings cloud to control the devices in your home.

FIGURE 5.

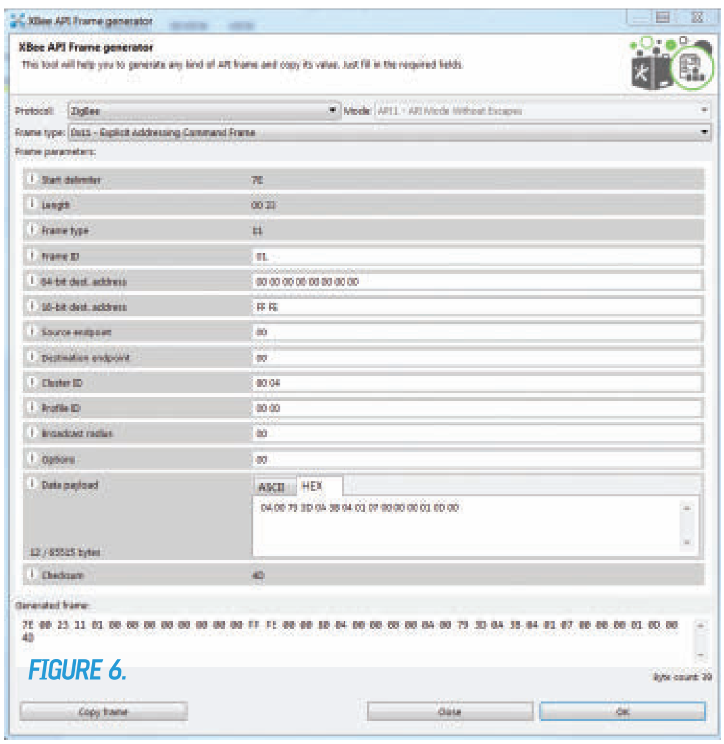


FIGURE 6.

Table 6

Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data Payload	0A	00	79	3D	0A	3B	04	01	07	00	00	00	01	0D	00

0A	Sequence number	Sequence number should be set to match the sequence number of the requesting packet.
00	Status	00 = Success
79 3D	Network address Two-byte little-endian	This is the four panel meter's two-byte network address in little-endian format. It must be read from the xBee during your firmware boot process just after it joins the network as this address is dynamic and will change.
0A	Length	Length in bytes of the simple descriptor that follows.
3B	End point report	This is the end point being reported.
04 01	Profile ID	Application Profile ID in little-endian 0x0104 is the profile ID for home automation.
07 00	Device type	Device type ID in little-endian format 0x0007 = Combined Interface; see page 51 of ZigBee Home Automation Profile.
00	Version	App dev version number.
00	Input cluster count	This end point has zero input clusters.
01	Output cluster count	This end point has one output cluster.
0D 00	Output cluster number	This is the listing of output clusters for this end point in little-endian format. In this case, we only have one 0x000D (analog output cluster).

What makes SmartThings unique is their cloud based integrated development environment (IDE) and the community of developers that work with it. The SmartThings IDE allows you to create custom device types and smart apps that can talk to your home automation devices.

If you create a custom device type or smart app, you can informally share them through **GitHub.com** or formally by going through the SmartThings certification process. Let's take a closer look at the SmartThings IDE.

First of all, here are a couple of links you will want to spend some time at:

- SmartThings IDE: **https://graph.api.smartthings.com**
- Documentation: **http://docs.smartthings.com/en/latest**

The documentation is good and getting better all the time. I encourage you to spend some time and read through it. The IDE allows you to build two types of applications: a "Device Handler" and "SmartApps." We will start with the device handler. You can think of it as a device driver for our IoT panel meter. Its primary function is to communicate with a device and parse out data into a common format that can be manipulated and used by a SmartApp. SmartApps are usually small in size and are intended to connect devices to the physical world.

For example, say we want to send wind speed information to one of the meters on our IoT panel meter. A SmartApp will allow us to pick a source of that wind speed and the meter number to send it to. The SmartApp will then continue to run in the SmartThings cloud and when the data source is updated, an event will fire and send a new value to the IoT panel meter.

Building a SmartThings Device Handler for the Four Panel Meter

I want to start off by saying this is not intended to be a screenshot by screenshot tutorial for building a device handler. You should be familiar with what a device handler is (also referred to as a custom device type) and how to navigate around in the SmartThings IDE. A good place to start is with the device type developer guide; you can walk through the quick start.

Here is a very simple SmartThings device handler for the four panel meter that will allow us to set the first meter's value. As you can see in **Figure 7**, the interface is simple with a slider and a text box showing the value being displayed on meter 1 of the IoT panel meter. When you slide your finger across the slider and remove it, the value of the slider will be sent to the IoT panel meter by sending a ZigBee write attribute command from the SmartThings hub. Here is the method from the

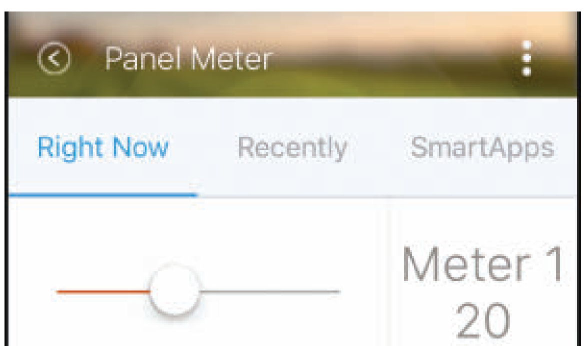
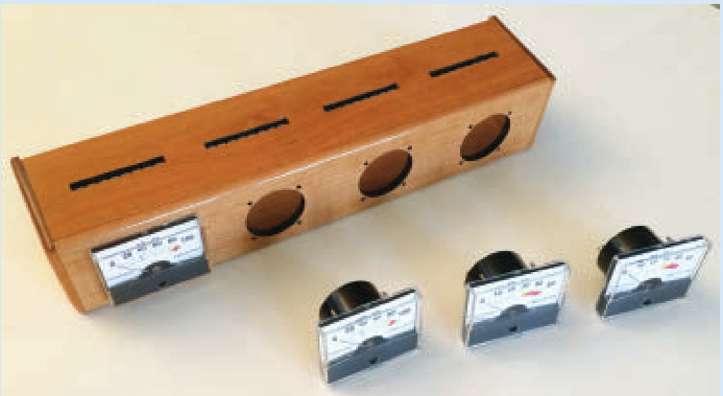


FIGURE 7. Simple Panel Meter device handler.



The IoT panel meter uses standard 50 µA analog meters.

Cherry wood box assembled and ready for hardware.

SmartThings device handler that sends the slider's value to the IoT panel meter:

```
private setMeter1Value(value){
     // Send event to update SmartThings
     sendEvent(name:"meter1_Value", value: value
     // Send to meter by writting value to
     // attribute 0x0055
     def endPointNumber = "0x38"
     def cluster =    "0x000D"
     def attribute = "0x0055"
     def dataType =     "0x39"
     def valueToSend =
Integer.toHexString(Float.floatToRawIntBits(value
as float))

     "st wattr 0x${device.deviceNetworkId}
     ${endPointNumber} ${cluster} ${attribute}
     ${dataType} {${valueToSend}}"
}
```

The parameters for the *st wattr* command should look familiar. You can see it is passed an end point number, cluster number, attribute number, data type, and finally the value as a floating point. This command is sent from the cloud to the SmartThings hub on your local network and then out the ZigBee radio to our IoT panel meter's network address.

Here is a link to the simple one panel device handler on GitHub: **https://github.com/JohnRucker/IoT-Panel-Meter/blob/master/devicetypes/iot-panel-meter/1-panel-meter-simple.src/1-panel-meter-simple.groovy**.

Building a Complete SmartThings Device Handler

This is the link to the full function device handler for the four panel meter "4-panel-meter.groovy:" **https://github.com/JohnRucker/IoT-Panel-Meter/blob/master/ devicetypes/iot-panel-meter/4-panel-meter.src/4-panel-meter.groovy**. You should install this device handler into your IDE by selecting the "Create New Device Handler" from the My Device Handlers tab of your IDE. Then select the "From Code" tab, and cut and paste all the text from this raw file. Select "Save" and "Publish for Me."

Figure 8 is a screenshot from this driver. This version of the device handler is a bit more complex than the single panel meter version. I not only added support for all four panel meters, but also added additional attributes and commands that will be used by SmartApps to send data to the IoT panel meter.

For the most part, this is a pretty standard SmartThings ZigBee device handler, but there are a couple of unique features I would like to focus on briefly.

FIGURE 8. Complete Panel Meter device handler.

At the beginning of this article, I discussed one of the attractions of an analog panel meter is the ability to tap on the faceplate to unstick the needle, verifying the displayed value is correct. To simulate this with our IoT panel meter, you push the red button on the back two times. This will cause the four panel meter's firmware to trigger an update from the SmartThings device handler. The parse method in the device handler will send new values for each panel meter updating their value. This event is triggered in the device hander by reporting the value of 0x00 for attribute 0x0000.

One of the attributes I added to this full function device handler was "meterX_Timeout." This attribute holds the time (in seconds) a panel meter will wait for new data (attribute 0x0500). Let's say meter 1 is reporting wind speed and we expect the wind speed data to be updated at least every five minutes. By setting the meter1_Timeout = 300 seconds, we are actually setting a timer in the firmware of the IoT panel

Resources

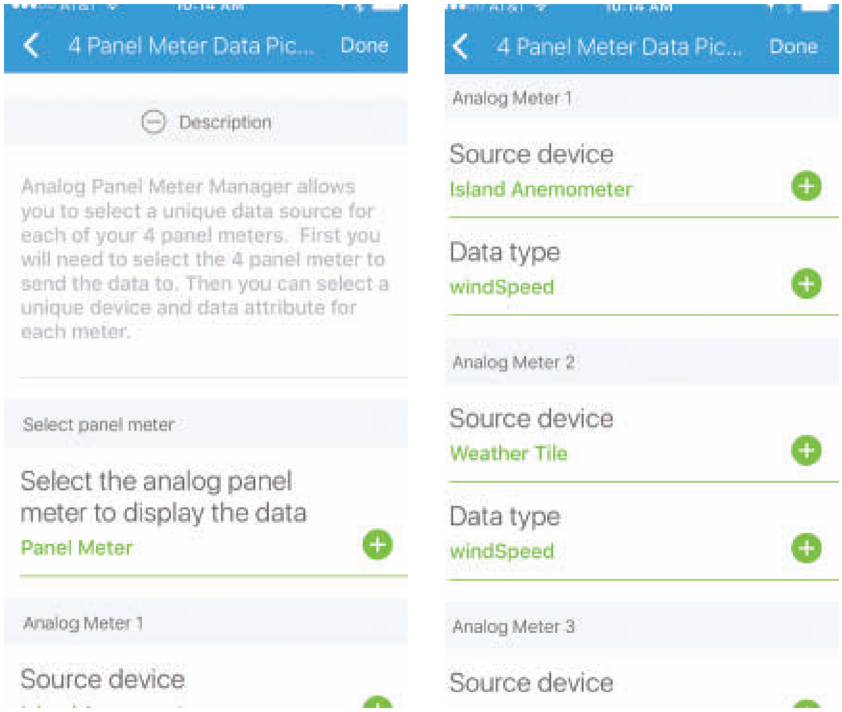
ZigBee Device Objects from the ZigBee Specification (ZDO)
www.zigbee.org/?wpdmdl=2168

ZigBee Cluster Library (ZCL)
https://docs.zigbee.org/zigbee-docs/dcn/07/docs-07-5366-02-0afg-zigbee-cluster-library-public-download-version.pdf

Digi xBee ZB Product Manual
http://ftp1.digi.com/support/documentation/90002002.pdf

meter. Now, the firmware will count the seconds between data updates it receives for panel 1. If the time is higher than 300 seconds, it will start to pulse the marquee's brightness letting us know the data is tardy for that panel meter.

So, now the IoT panel meter knows if the data being displayed is current and if it isn't, it has a way to signal that to the user and request an update from the data source.



FIGURES 9 and 10. IoT four panel meter SmartApp smartphone screens.

Selecting Data to Display with a SmartApp

At this point, we have our IoT panel meter connected to our device handler, and we can use our smartphone to manually set the values for each panel meter. If we want to use something other than our finger to change the values of each panel meter, we need a SmartApp. SmartApps allow us to consume events that fire in the SmartThings cloud and act on them. For example, let's say you have a temperature probe connected to your SmartThings hub. Every time a new temperature value is reported to the hub, an event fires within the SmartThings cloud. A SmartApp allows us to consume that event and do something with the value when it fires (is updated). That is how we grab data and send it to our IoT panel meter.

A good place to start with SmartApps is the online tutorial from SmartThings, "Writing Your First SmartApp." They do a good job of illustrating how you select devices and subscribe to their events. Once you have a good understanding of what a SmartApp is, take a look at the IoT four panel meter data picker SmartApp code. **Figure 9** and **Figure 10** are screenshots from my smartphone to give you a feel for what the SmartApp looks like.

As you can see, the SmartApp allows you to pick a panel meter to send the data to and data sources to pull the data from. The data source can be other ZigBee devices, Z-Wave devices, Wi-Fi devices connected to SmartThings, or even Internet hosted data in JSON format.

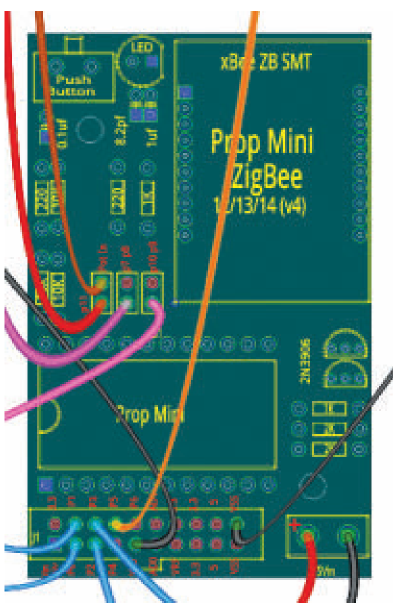


FIGURE 11.

Hardware

Figure 11 shows a Fritzing wiring diagram of my Prop Mini ZigBee prototype printed circuit board (PCB). This PCB is handy for prototyping ZigBee based projects as it takes care of all the wiring for the xBee radio and the Propeller Mini (see schematic in **Figure 14**). It also has some common components, like a couple of PNP transistors for level shifting, an RC network for measuring resistance, as well as a couple of pins pulled high for connection to external pushbuttons.

Figure 12 shows the connections for the four analog panel meters, four marquees, buzzer, and the lit pushbutton for the IoT sourced panel meter. The wires are soldered to the PCB making sturdy connections, allowing the whole thing to be tucked neatly into the wood box of the panel meter.

Figure 13 shows a picture of the Prop Mini ZigBee protoboard ready for connection to the analog panel meters and the marquees daisychained at the top of the wood box. As mentioned, **Figure 14** is a schematic of the Prop Mini

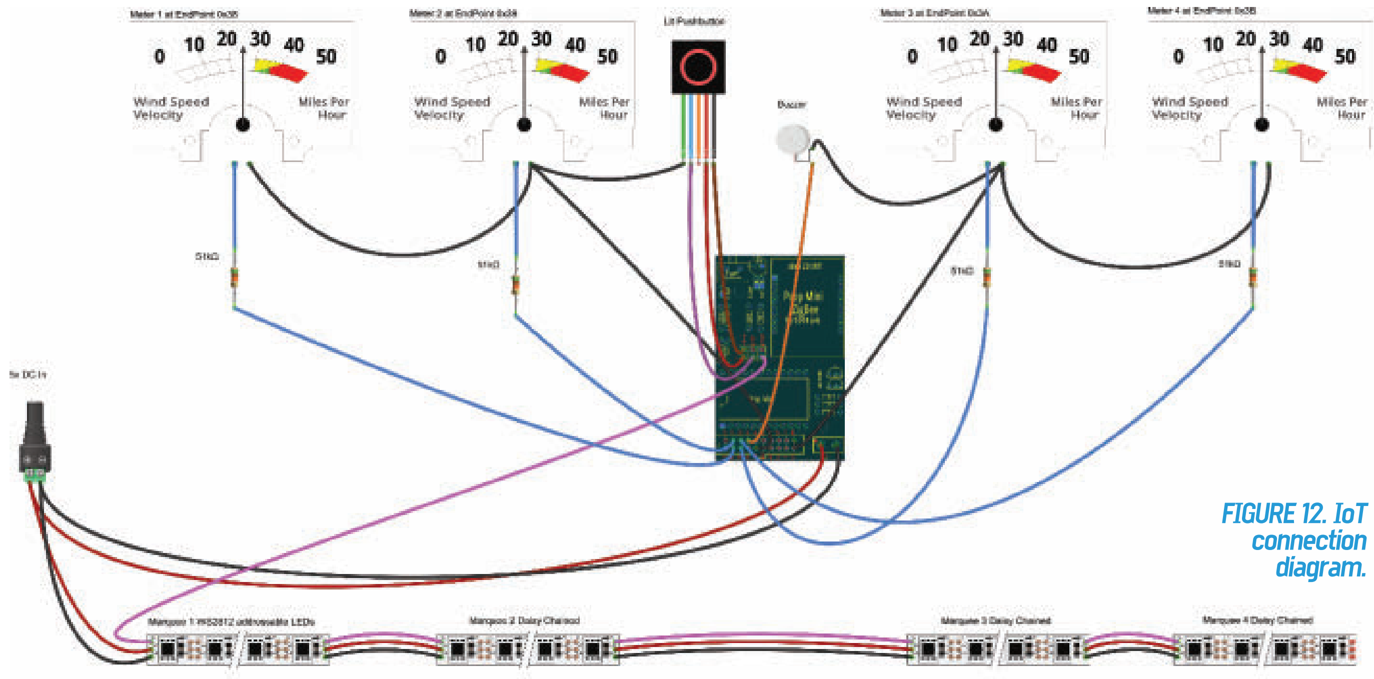


FIGURE 12. IoT connection diagram.

ZigBee PCB. Note there are five headers on the board. The header at J1.1 and J1.2 has several general-purpose I/O pins as well as power. The header labeled “5Vin” is where you can connect a 5V DC power source.

Note this is connected to the 5V out of the Propeller Mini. This is okay as the Propeller will use the 5V to source its onboard 3.3V regulator.

The header labeled “Out” has the output of two PNP transistors that are driven by pings 9 and 10 of the Propeller. The header labeled “Pot” has an RC network connected to P11 and is where you can connect a photoresistor or other variable resistance to be read by the Propeller. Finally, the header labeled “In” has two 10K resistors pulling pins 8 and 7 high for connecting a pushbutton.

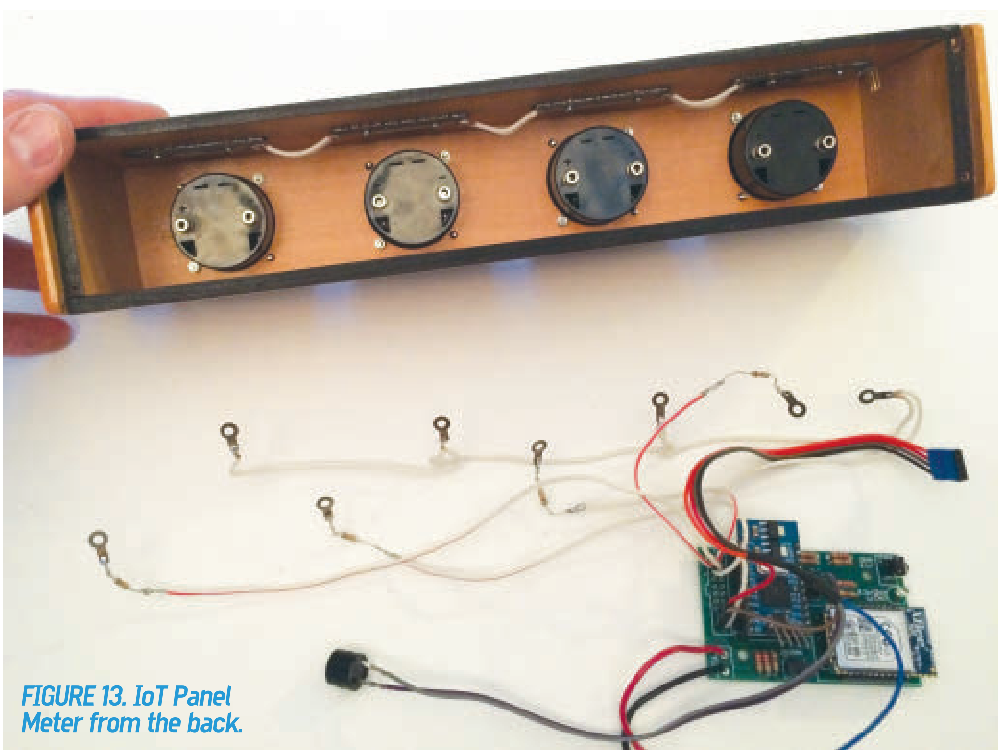


FIGURE 13. IoT Panel Meter from the back.

Microcontroller Firmware

As I mentioned earlier, the microcontroller I’m using for my IoT sourced panel meter is a Parallax Propeller. The firmware is written in Spin which is the Propeller’s native language. I have commented the code so it should be pretty self-explanatory if you’re an experienced Spin programmer. If not, Spin is similar to C+, so it shouldn’t be too difficult to get an idea of what is going on if you have a programming background.

The firmware for the Propeller can be found at the article link and at my GitHub site. Here, you will find several Spin object files and two Spin programs: *PanelMeterSetup.spin* and *ZigBeePanelMeter.Spin*. The

PanelMeterSetup.spin program allows you to set up, test, and calibrate new meters.

Each panel meter will need to be calibrated once to make up for inaccuracies in the current-limiting resistor driving that panel meter.

Load and run the *PanelMeterSetup.spin* program and follow the menu to calibrate each panel meter. This process will create new DAT settings for the meterMaxFRQ, meterScaleMax, and meterScaleMin variables as shown in the DAT section in **Figure 15**.

Once you have the new calibration settings, you should update the values in the *PanelMeter.spin* object file and save the file with your new calibration settings. The *PanelMeter.spin* file is a Spin object that is used by both *PanelMeterSetup.spin* and *ZigBeePanelMeter.Spin* to drive the four panel meters.

Now that the calibration is complete, you can load the main program *ZigBeePanelMeter.spin* into the EEPROM of your Propeller Mini and boot it up. Connect the Parallax Serial Terminal (PST) to your Prop plug at 250,000 baud to see a running log of the boot process. The Prop Mini will boot up, connect to the xBee, and wait for the xBee to connect to your SmartThings hub. At this point, you should have already installed, saved, and "Publish for me" the "4-panel-meter" device handler (see the previous *Building a Complete SmartThings Device Handler* section).

To allow the Propeller Mini to join the SmartThings network, use your Smartphone and open the network for joining by adding a new device. Once the network has been opened for joining, you should see packets starting to flow in the PST log. The SmartThings cloud will identify the Propeller Mini as an IoT panel meter based on the footprint in the "4-panel-meter" device handler; it should appear on your smartphone. You can open the "ZigBeePanelMeter" on your smartphone and use your finger to change the slider for each analog panel meter. Verify the needle moves to the correct location on the meter face for the value selected.

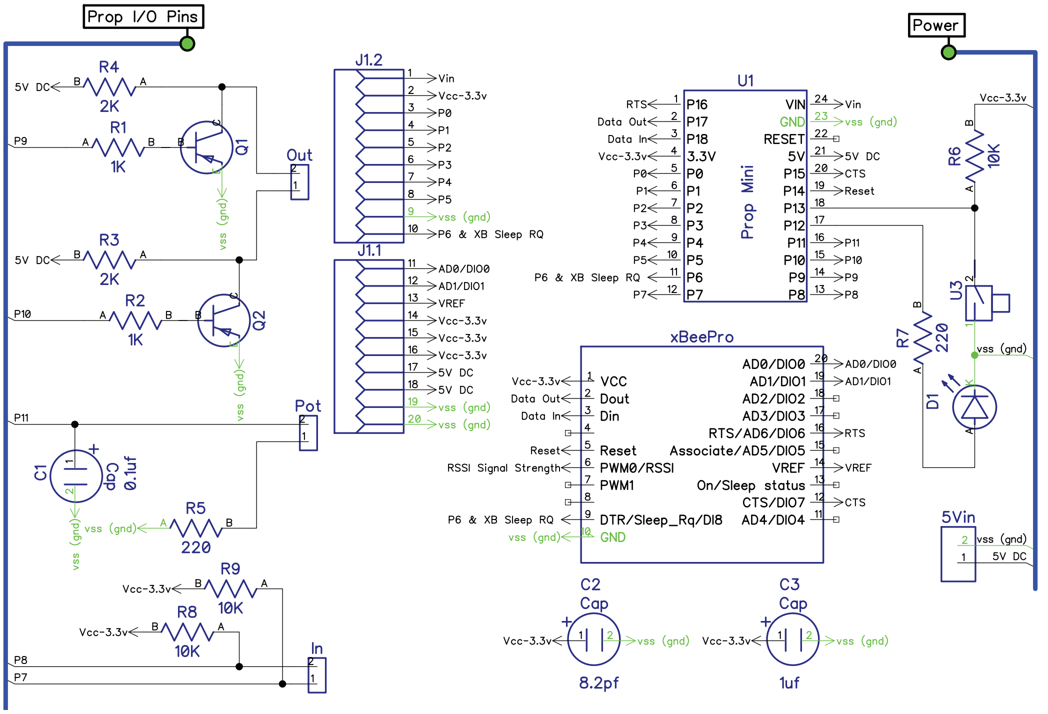


FIGURE 14. Prop Mini ZigBee PCB schematic.

```
DAT
        meterValue      Long    0[mCount]                ' Value to display on meter
        meterAlarmLow   Long    -1, -1, -1, -1           ' Low value that will cause an alarm
        meterAlarmWarn  Long    20, 20, 80, 80           ' Warning value will turn marquee Yellow
        meterAlarmHigh  Long    30, 35, 90, 90           ' High value that will trun marquee Red
        meterAlarmMajor Long    40, 40, 100, 100         ' Value that will cause flashing Red
        alarmMute       Long    False[mCount]            ' Used to mute active alarm




        meterMaxFRQ     Long    223, 215, 213, 214       ' The FRQ value that will peg the needle to max tailored to each meter
        meterScaleMax   Long     50,  50, 100, 100       ' Meter scale max. If set to 50 and meterScaleMin - 0, values from 0 to 50 are displayed
        meterScaleMin   Long      0,   0,   0,   0       ' Meter scale min can be a negative number

        AlertOnNewValue Long    True[mCount]             ' True if you want new values to cause a wipe right or left on the marquee
        MarqueeShowNum  Long    0[mCount]                ' 1=Stop, 2=alarm minor, 3=alarm, 4=major alarm, 5=flash bright, 6=chase right, 7=chase
        MarqueeCount    Long    mCount                   ' Set MarqueeCount so other objects can see mCount

```

FIGURE 15.

Note: When a meter receives data, you will hear a soft click sound for each valid data packet and see the needle move.

If you haven't done so already, the next step is to install the 4-panel-meter-data-picker.groovy SmartApp. Cut and paste the raw SmartApp file (at **https://raw.githubusercontent.com/JohnRucker/IoT-Panel-Meter/master/smartapps/iot-panel-meter/4-panel-meter-data-picker.src/4-panel-meter-data-picker.groovy**) into the SmartThings IDE (under the My SmartApps tab) and "Publish for me." This app will allow you to select data sources for your IoT meter and can also be a template for future SmartApps. Please feel free to share your SmartApps with the SmartThings community as that is what makes SmartThings so unique. They have a very rich supportive online developer community.

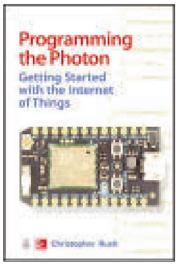
I hope you find this information useful and I would enjoy hearing what you do with it! Please feel free to contact me on the SmartThings forum or in this article's discussion page at the link provided. **NV**

For complete product details, visit us online!!

The Nuts & Volts WEBSTORE

GREAT FOR DIYers!

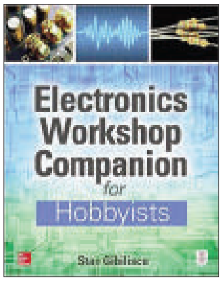
Programming the Photon: Getting Started with the Internet of Things
by Christopher Rush



Quickly learn to construct your own electronics devices and control them over the Internet with help from this DIY guide. This guide features clear explanations and step-by-step examples that use inexpensive, easy-to-find components. Discover how to connect to Wi-Fi networks, attach hardware to I/O ports, write custom programs, and work from the cloud. You will learn how to troubleshoot and tweak even interface with social media sites!

$20.00

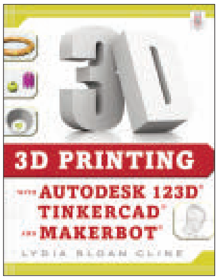
Electronics Workshop Companion for Hobbyists
by Stan Gibilisco



In this practical guide, electronics expert Stan Gibilisco shows you, step by step, how to set up a home workshop so you can invent, design, build, test, and repair electronic circuits and gadgets. Electronics Workshop Companion for Hobbyists provides tips for constructing your workbench and stocking it with the tools, components, and test equipment you'll need. Clear illustrations and interesting do-it-yourself experiments are included throughout this hands-on resource.

$25.00

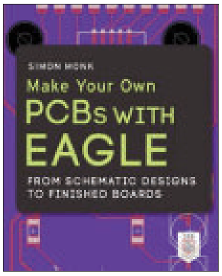
3D Printing with Autodesk 123D, Tinkercad, and MakerBot
by Lydia Cline



Are you ready to join the new industrial revolution? *3D Printing with Autodesk 123D, Tinkercad, and MakerBot* reveals how to turn your ideas into physical products that you can use or sell! You'll learn how to operate powerful, free software from Autodesk and bring your creations to life with the MakerBot (a leading consumer printer) or an online service bureau.

$30.00

Make Your Own PCBs with EAGLE
by Eric Kleinert

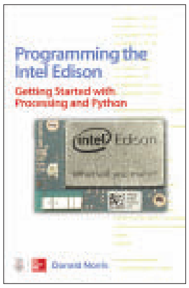


Featuring detailed illustrations and step-by-step instructions, *Make Your Own PCBs with EAGLE* leads you through the process of designing a schematic and transforming it into a PCB layout. You'll then move on to fabrication via the generation of standard Gerber files for submission to a PCB manufacturing service. This practical guide offers an accessible, logical way to learn EAGLE and start producing PCBs as quickly as possible.

$30.00

Programming the Intel Edison: Getting Started with Processing and Python
by Donald Norris

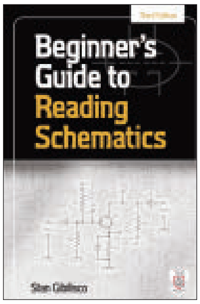
Learn To Easily Create Robotic, IoT, and Wearable Electronic Gadgets!



Discover how to set up components, connect your PC or Mac, build Python applications, and use USB, WiFi, and Bluetooth connections. Start-to-finish example projects include a motor controller, home temperature system, robotic car, and wearable hospital alert sensor..

$20.00

Beginner's Guide to Reading Schematics, 3E
by Stan Gibilisco

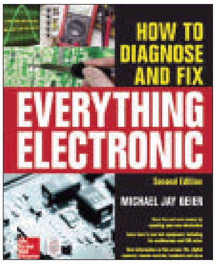


Navigate the roadmaps of simple electronic circuits and complex systems with help from an experienced engineer. With all-new art and demo circuits you can build, this hands-on, illustrated guide explains how to understand and create high-precision electronics diagrams. Find out how to identify parts and connections, decipher element ratings, and apply diagram-based information in your own projects.

$25.00

How to Diagnose and Fix Everything Electronic
by Michael Jay Geier

A Fully Revised Guide to Electronics

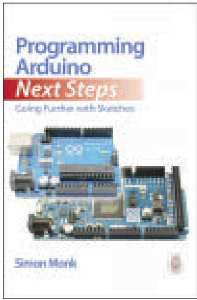


Repair all kinds of electrical products, from modern digital gadgets to analog antiques, with help from this updated book. The Second Edition offers expert insights, case studies, and step-by-step instruction from a lifelong electronics guru. Discover how to assemble your workbench, use the latest test equipment, zero in on and replace dead components, and handle reassembly. **$24.95**

Programming PICs in Basic
by Chuck Hellebuyck

If you wanted to learn how to program microcontrollers, then you've found the right book! Microchip PIC microcontrollers are being designed into electronics throughout the world and none is more popular than the eight-pin version. Now the home hobbyist can create projects with these little microcontrollers using a low cost development tool called the CHIPAXE system and the Basic software language. Chuck Hellebuyck introduces how to use this development setup to build useful projects with an eight-pin PIC12F683 microcontroller. **$14.95**

Programming Arduino Next Steps: Going Further with Sketches
by Simon Monk

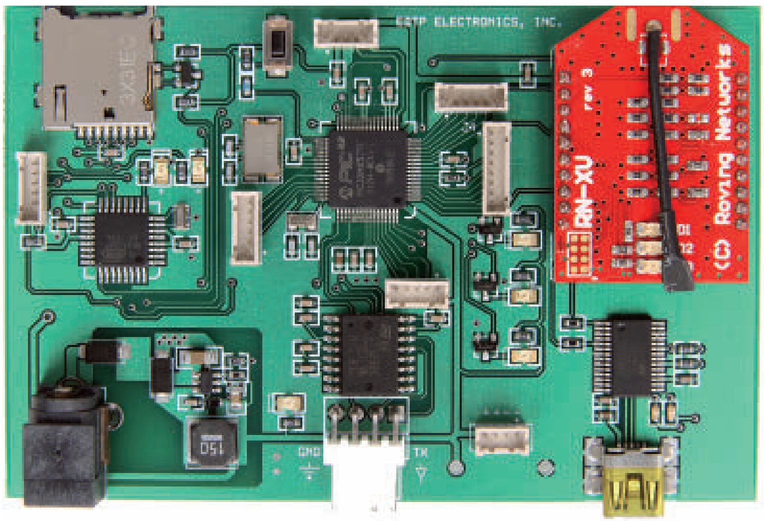


In this practical guide, electronics guru Simon Monk takes you under the hood of Arduino and reveals professional programming secrets. Also shows you how to use interrupts, manage memory, program for the Internet, maximize serial communications, perform digital signal processing, and much more. All of the 75+ example sketches featured in the book are available for download. **$20.00**

■ BY FRED EADY

Let Your PIC Push the Buttons in this IoT Device

In the previous installment of Design Cycle, we took a microscopic look at the A40 Mesh Networking Alarm Controller. Now that we understand how the A40 alarm system operates, it's time to do what we do best. This month, we're going to create an IoT (Internet of Things) device whose sole job is to monitor and control an A40 network. The PIC32MX575F512H hardware we have designed and constructed is well suited for the task. So, let's get crackin'.

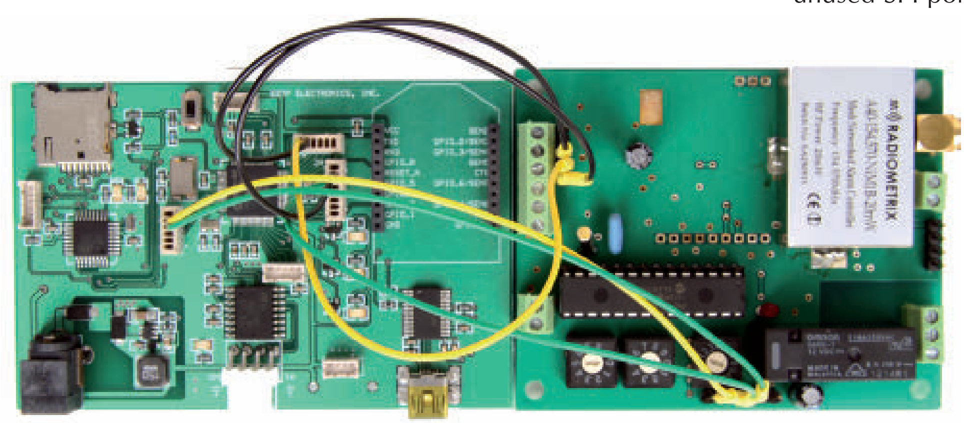


■ PHOTO 1. This hardware was designed to be flexible. Almost all of the PIC32MX575F512H's GPIO pins that are not already in use are exposed for external connections.

IoT Hardware Integration

The first order of business is to add some DF-13 male connectors to our IoT hardware. As you can see in **Photo 1**, DF-13 hardware has been added at J4 and the remaining DF-13 pad areas that were bare in our last discussion. The DF-13 connectors were added to allow us access to the PIC32MX575F512H's external interrupt pins, GPIO, and unused UART. The A40 mesh networking alarm controller's GPIO logic level is 3.3 volts, which means that we can directly interface the A40 controller's I/O to the PIC32MX575F512H. As you can see in **Photo 2**, we can actually jumper the A40 alarm controller and our PIC32MX575F512H hardware together using the DF-13 interfaces.

We have the luxury of choosing between a pair of unused PIC32MX575F512H serial ports. Serial ports 3 and 6 are unused and pinned out on DF-13 connector J4. Using either of them will trash the unused SPI portal, but it isn't on our agenda anyway. The alarm controller doubles its GPIO terminations using both screw terminals and standard 2.54 mm pitch male headers. For now, we will connect the PIC32MX575F512H hardware to the A40 hardware using the male headers that are depicted in **Figure 1**.



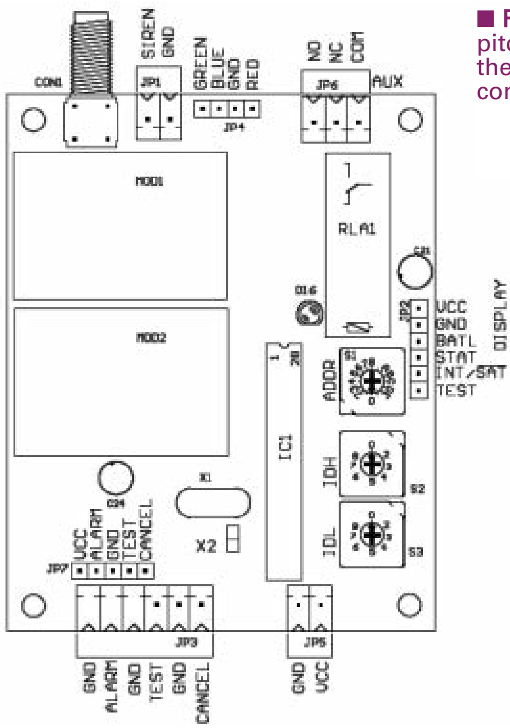
■ PHOTO 2. With just six connections, the A40 mesh networking alarm controller module now has access to a Wi-Fi radio, a microSD card, an RS-232 port, and a USB port.

Making the Connections — BATL

The A40 mesh alarm controller favors blinking LEDs at specified intervals to notate status.

Since we have a microcontroller in the mix, we can pick up on these alternating signals and show their meaning on a graphical display.

Low battery status is reported via the BATL signal, which pulses for 100 mS every four seconds. Obviously, we can't wait around doing nothing for four seconds to see if the battery is low. So, we'll use the PIC32MX575F512H's external interrupt subsystem to handle any of the low battery situations. According to **Schematic 1**, we have all of the external interrupt pins available with the exception of INT2, which is assigned to RS-232 receive duty. So, let's assign the BATL signal duty to INT4.



■ **FIGURE 1.** The standard 2.54 mm pitch male headers make interfacing to the A40 mesh networking alarm controller quick and easy.

Making the Connections — STAT

Alarm and network status is reported every 32 seconds using 3.3 volt serial signaling on the A40 controller's STAT pin. The stat signal is actually a 9600 baud inverted RS-232 signal that is perfect for our PIC32MX575F512H UART. In this case, we only need to receive from the alarm controller. Another look at **Schematic 1** reveals that we can save the unused SPI portal by receiving the STAT data with UART6. So it is written, so shall it be done.

Making the Connections — INT

The INT output signal is used to alert us to an upcoming STAT transmission. We can use this signal as an "I'm alive" indicator. We can also use the INT pulse as a signal to retrieve the STAT data. Again, we don't want to be standing on the microcontroller street corner waiting for that 1 mS periodic INT pulse. So, let's assign the keep-alive watch to the PIC32MX575F512H's INT3 pin.

Making the Connections — TEST and CANCEL

We only need to connect a couple of A40 alarm controller input signals: TEST and CANCEL. These inputs are expecting a physical pushbutton to be attached to the alarm controller's input interface. Our PIC32MX575F512H can easily be programmed to mimic physical pushbuttons. The term for this physical switch emulation is mechatronics. According to **Schematic 1**, we have some open output pins in PORTB. With that, let's assign RB0 to TEST and RB1 to CANCEL.

Configuring the PIC32MX575F512H

Now that we have all of our physical connections identified, let's code them in. The first thing we must do on the coding side is prepare our PIC. Our design includes an 8 MHz crystal to drive the CPU clock. With the help of the PIC32MX575F512H's PLL, we can clock the CPU and its peripherals at 80 MHz. The trick is to divide the incoming 8 MHz clock signal by two. The reason for this is that the PIC32MX575F512H's PLL wants to see an incoming clock of 4 MHz.

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/June2016_DesignCycle-Portable-RF-Mesh-Alarm-System-PIC-IoT**.

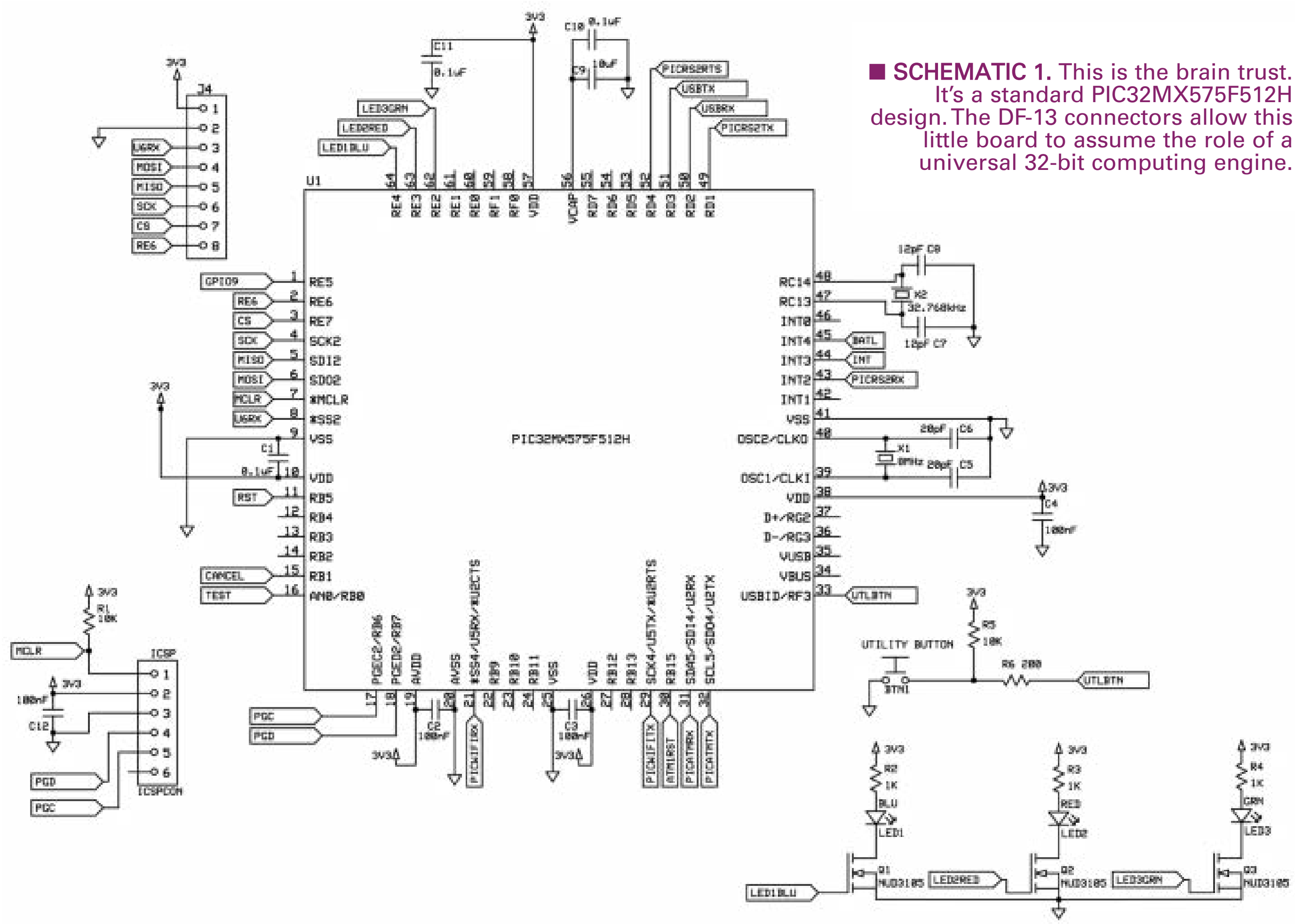
Once the PLL has control of the incoming clock signal, we can instruct it to multiply the clock by 20. This results in our 80 MHz CPU clock. Using the configuration words, the same 80 MHz signal can also be routed to the peripherals and GPIO subsystem. Here is the configuration fuse code layout:

```
//**************************************************
//* CONFIGURATION WORDS
//**************************************************
     #pragma config FNOSC = PRIPLL
       // Oscillator Selection
     #pragma config POSCMOD  = HS
       // Primary Oscillator
     #pragma config FPLLIDIV = DIV_2
       // PLL Input Divider
     #pragma config FPLLMUL = MUL_20
       // PLL Multiplier 20, 2
     #pragma config FPLLODIV = DIV_1
       // PLL Output Divider
     #pragma config FPBDIV  = DIV_1
       // Peripheral Clock divisor
     #pragma config FWDTEN  = OFF
       // Watchdog Timer
     #pragma config WDTPS = PS1
       // Watchdog Timer Postscale
     #pragma config FCKSM = CSDCMD
       // Clock Switching & Fail Safe Clock
       // Monitor
     #pragma config OSCIOFNC = OFF
       // CLKO Enable
     #pragma config POSCMOD = HS             1
       // Primary Oscillator
     #pragma config IESO = OFF
       // Internal/External Switch-over
     #pragma config FSOSCEN  = OFF
       // Secondary Oscillator Enable (KLO
       // was off)
     #pragma config CP = OFF
       // Code Protect
     #pragma config BWP = OFF
       // Boot Flash Write Protect
     #pragma config PWP = OFF
       // Program Flash Write Protect
     #pragma config ICESEL = ICS_PGx2
       // ICE/ICD Comm Channel Select
```

Pardon My Interrupts

We have a couple of periodic pulses to monitor. Although the PIC32MX575F512H is a master at following our instructions, it doesn't have a clue as to when the BATL or INT pulses will come knocking. So, we'll lay a couple of "traps" for them under the guise of external interrupts. Setting up the external interrupts is a straightforward affair.

At power-up, the INT3 and INT4 pins are configured



■ **SCHEMATIC 1.** This is the brain trust. It's a standard PIC32MX575F512H design. The DF-13 connectors allow this little board to assume the role of a universal 32-bit computing engine.

as inputs by default. Our job is to instruct the PIC32MX575F512H to look for low to high logic transitions on either of the external interrupt pins. We do this by enabling the external interrupts and coding a couple of external interrupt handlers. Here is the external interrupt setup code for INT3:

```
//Initialize INT3
INTCONSET = _INTCON_INT3EP_MASK;
   //trigger on rising edge
//priority 3 sub priority 0 - 00011111 bits
  <28:26> sub bits <24:25>
IPC3SET = 0x0C000000; //0000 1100 0000 0000
0000 0000 0000 0000
IEC0SET = 0x00008000;
IFS0CLR = 0x00008000; //0000 0000 0000 0000
1000 0000 0000 0000
```

And INT4:

```
//Initialize INT4
INTCONSET = _INTCON_INT4EP_MASK;
   //trigger on rising edge
//priority 4 sub priority 0 - 00011111 bits
  <28:26> sub bits <24:25>
IPC4SET = 0x10000000; //0001 0000 0000 0000
  0000 0000 0000 0000
IEC0SET = 0x00080000;
IFS0CLR = 0x00080000; //0000 0000 0000 1000
  0000 0000 0000 0000
```

Setting up the external interrupts is just like setting any other PIC32MX575F512H interrupt. We must choose and set an interrupt priority (IPCx). The priority levels for the PIC32MX575F512H run from level 7 (highest priority) to level 1 (lowest priority). As it stands, we have chosen to put the low battery interrupt at a higher priority than the keep-alive interrupt. That means that our low battery interrupt will always fire before the keep-alive interrupt. Setting the correct IECx bit will enable the associated interrupt. The IFSx bits are interrupt flags that we must clear after each interrupt fires.

The BATL and INT interrupt handler code doesn't have to be as tight in this application as the pulses that trigger the external interrupts are exceptionally lengthy. So, we can dibble dabble inside of the interrupt handler for a number of milliseconds if we desire. Because we are great coders and engineers, we will do it right and get in and out of the interrupt handlers as quickly as possible. However, to avoid premature retriggering, we will make sure the BATL and INT logic levels are low before we exit the interrupt handler. Here's the interrupt handler for INT3 (INT signal):

```
//**********************************************
// INT3 interrupt handler
// it is set at priority level 3 with software
// context saving
//**********************************************
void __ISR(_EXTERNAL_3_VECTOR, IPL3SOFT)
INT_Handler(void)
{
       grnLED_On;
       flags.fint = 1;
       tdelayms(10);//wait for signal to return
                    //to logical low level
       IFS0CLR = 0x00008000; //0000 0000 0000
       0000 1000 0000 0000 0000
}
```

And INT4 (BATL signal):

```
//**********************************************
// INT4 interrupt handler
// it is set at priority level 4 with software
// context saving
//**********************************************
void __ISR(_EXTERNAL_4_VECTOR, IPL4SOFT)
BATL_Handler(void)
{
       bluLED_On;
       flags.fbatl = 1;
       tdelayms(150);
         //wait for signal to return to logical
         //low level
       IFS0CLR = 0x00080000; //0000 0000 0000
         1000 0000 0000 0000 0000
}
```

As you can see in the interrupt handler code, we've illuminated the green and blue LEDs when their respective interrupt fires. We've also set a flag bit within each interrupt handler. We will use the flag bits to steer events within the main application. Here's the code that sets up the flag bits:

```
typedef struct
{
       BYTE   fint:1;
       BYTE   fbatl:1;
}FFLAGS;
FFLAGS flags;
```

Now that we can sense the INT pulse, we will need to act on it by receiving status data. So, let's go ahead and set up UART6 as a receive-only serial port. We will begin this process by defining and allocating a receive ring buffer:

```
//**********************************************
//* USART RECEIVE BUFFERS SETUP
//* 2,4,8,16,32,64,128 or 256 bytes
//**********************************************
#define USART6_RX_BUFFER_SIZE  256
#define USART6_RX_BUFFER_MASK (
  USART6_RX_BUFFER_SIZE - 1 )
BYTE USART6_RxBuf[USART6_RX_BUFFER_SIZE];
BYTE USART6_RxTail;
BYTE USART6_RxHead;
```

We're also going to allocate a ring buffer for UART1. We can use this UART for debugging purposes as it is dedicated to the FTDI based USB-to-serial port. Enabling UART1 allows us to connect to a terminal emulator on a PC via the FTDI-controlled serial port. The UART1 ring buffer code is identical to the UART6 code:

```
#define USART1_RX_BUFFER_SIZE  256
#define USART1_RX_BUFFER_MASK (
  USART1_RX_BUFFER_SIZE - 1 )
BYTE USART1_RxBuf[USART1_RX_BUFFER_SIZE];
```

```
BYTE USART1_RxTail;
BYTE USART1_RxHead;
```

With the receive ring buffer space allocated in the PIC32MX575F512H's SRAM, we can now focus on actually enabling the UARTs and setting up their receive interrupts and interrupt handlers:

```
    //Initialize UART1
    U1BRG = 520; // Set Baud rate 9600
    U1STA = 0;
    U1MODE = 0x00008000;
       // Enable UART for 8-none-1
    U1STASET = 0x00001400;
       // Enable Transmit and Receive
    //priority 2 sub priority 3 - 00011111 bits
      <4:2> sub bits <0:1>
    IPC6SET = 0x0000000B;
    IEC0SET = 0x08000000;
    IFS0CLR = 0x08000000;
       // flush receive buffer
    USART1_RxTail = 0x00;
    USART1_RxHead = 0x00;

    //Initialize UART6
    U6BRG = 520; // Set Baud rate 9600
    U6STA = 0;
    U6MODESET = 0x00008000;
       // disable UART for 8-none-1
    U6STASET = 0x00001400;
       // Enable Transmit and Receive
     //priority 2 sub priority 3 - 00011111 bits
      <20:18> sub bits <17:16>
    IPC12SET = 0x000B0000;//0000 0000 0000 1011
      0000 0000 0000 0000
    IEC2SET = 0x00000080;
    IFS2CLR = 0x00000080; //0000 0000 0000 0000
      0000 0000 1000 0000
    //flush receive buffer
    USART6_RxTail = 0x00;
    USART6_RxHead = 0x00;
```

Another plus associated with enabling UART1 is the ability to use it as a command input. If we choose to, we can write code that would allow us to manipulate the graphic display from the PC's keyboard. We could also write code to interface our PC to the A40 controller module. As it stands right now, we will only use UART1 for reporting or debugging purposes.

As you can see in the UART setup code, UART1 and UART6 are configured to run at 9600 baud with eight-bit frames, 1 STOP bit, and no parity. In that UART1 will not be operating simultaneously with UART6, the interrupt priorities are not important at this point. You can get a better idea of the UART1 and UART6 bit settings by taking a look at the interrupt section of the PIC32MX575F512H datasheet.

The UART interrupt handlers are identical with the exception of the UART that is called out in the code. Basically, the UART receive interrupt handler retrieves the incoming byte from the associated UART and pushes the received byte into the circular receive buffer at the head pointer location. Let's look at the UART6 receive interrupt handler code:


```
//***********************************************
// UART 6 interrupt handler
// it is set at priority level 2 with software
// context saving
//***********************************************
void __ISR(_UART_6_VECTOR, IPL2SOFT)
  IntUart6Handler(void)
{
       BYTE data,tmphead,tmptail;
       //get the incoming byte
       data = U6RXREG;
       //calculate buffer index
       tmphead = ( USART6_RxHead + 1 ) &
         USART6_RX_BUFFER_MASK;
       //store new index
       USART6_RxHead = tmphead;
       if ( tmphead == USART6_RxTail )
       {
       // ERROR! Receive buffer overflow
       }
       //store received data in buffer
       USART6_RxBuf[tmphead] = data;
       //0000 0000 0000 0000 0000 0000 1000 0000
       IFS2CLR = 0x00000080;
}
```

The logic behind the code in the interrupt handler is revealed in the XC32 user's guide. The UART interrupt handler pushes received bytes onto the receive buffer at the head. The *recvchar* functions pull data off of the receive ring buffer at the tail location:

```
//***********************************************
//*    USART6 Receive Character Function
//***********************************************
BYTE recvchar6(void)
{
  BYTE tmptail6;
  //calculate buffer index
  tmptail6 = ( USART6_RxTail + 1 ) &
   USART6_RX_BUFFER_MASK;
  //store new index
  USART6_RxTail = tmptail6;
  //return the retrieved byte
  return USART6_RxBuf[tmptail6];
}
```

Although the reception of incoming serial data is handled by interrupts and receive ring buffers, we must poll the status of the ring buffers to ascertain if any characters are stored in the buffers and ready for extraction. A very simple line of code accomplishes this:

```
//***********************************************
//*    USART6 Character Waiting Function
//***********************************************
BYTE CharInQueue6(void)
{
  return(USART6_RxHead != USART6_RxTail);
}
```

Necessary Preparations

The PIC32MX575F512H needs some attention before we can turn it loose on our application code. We need to set up the PIC32 for maximum performance. Once that's done, our initialization code must configure the PIC32MX575F512H's GPIO pins.

The code to perform the aforementioned tasks is not

complicated as you can see here:

```
    // PBCLK - already set to SYSCLK/1 via
    // config settings
    // Data Memory SRAM wait states: Default
    // Setting = 1; set it to 0
    BMXCONbits.BMXWSDRM = 0;
    // Flash PM Wait States: MX Flash runs at 2
    // wait states @ 80 MHz
    CHECONbits.PFMWS = 2;
    // Prefetch-cache: Enable prefetch for
    // cacheable PFM instructions
    CHECONbits.PREFEN = 1;
    // JTAG: Disable on PORTA
    DDPCONbits.JTAGEN = 0;
    AD1PCFG = 0xFFFF;
    TRISBCLR = 0xC023; //1100 0000 0010 0011
    TRISDCLR = 0x000A; //0000 0000 0000 1010
    TRISECLR = 0x003C; //0000 0000 0011 1100
    TRISFCLR = 0x0020; //0000 0000 0010 0000
```

The two “killers” when it comes to “why isn’t this working” are the JTAG and analog settings. If you don’t disable JTAG, the PIC32MX575F512H’s JTAG functions have priority over the pins they occupy. Failing to disable analog inputs on your desired digital pins will result in outputs seeming not to work. So, JTAG and all analogs (AD1PCFG) are disabled in our *init* function.

Since all of the PIC32MX575F512H’s GPIO pins power up as inputs, we need only declare and configure the output pins. The TRIS bits that are cleared associate with the GPIO pins that are to be configured as outputs. For instance, GPIO pin RB0 was designated as our TEST pushbutton replacement pin. Note that it and the adjacent CANCEL pushbutton replacement pin’s TRIS bit is cleared. Thus, RB0 and RB1 are configured as output pins.

Application Code

We’ve written enough infrastructure code to begin assembling the application code. Let’s put the external interrupts for INT, BATL, and UART6 to the test:

```
int main (void)
{
    BYTE bitein,i;
      init();
      do{
             if(flags.fint)
             {
                    if(CharInQueue6())
                    {
                           for(i=0;i<16;i++)
                           {
if(CharInQueue6())
                                  {
pktBuf[i] = recvchar6();
printf("%c",pktBuf[i]);
                                  }
                           }
                    }
                    flags.fint = 0;
             }
```

Our application code snippet initializes the PIC32MX575F512H’s subsystems and variables. When the INT interrupt fires, we set the *flag fint*. We use the logical state of the *flag fint* to determine whether or not to retrieve incoming status data via UART6. If data is retrieved, it is echoed out to the FTDI USB portal via UART1.

The low battery application snippet is far less complex. If the mesh networking alarm controller signals a battery voltage below four volts, the BATL interrupt fires. The result is that every four seconds, the red LED on the PIC32MX575F512H board toggles between dark and light:

```
      if(flags.fbatl)
      {
             redLED_Tog;
             flags.fbatl = 0;
      }
```

What’s Next?

I have supplied the A40 alarm controller code package at the article link. You can also consult back issues of *Nuts & Volts* to gain access to the complete plans for building the PIC32MX575F512H hardware. Next time, we’ll weave a color touch panel into our A40 mesh networking alarm controller based IoT device. **NV**

How Wireless Signals Propagate

Radio waves are a combination of electric and magnetic fields that travel together through free space, from transmitting antenna to receiving antenna. However, it is not as simple as that. For example, lower frequency signals like those of AM radio stations (535 to 1,705 kHz) travel along near the ground. High frequency (3-30 MHz) signals like shortwave and amateur radio are launched into the sky and are refracted (not reflected) by the ionosphere back to earth so are able to travel long distances. VHF, UHF, microwave, and millimeter wave signals (30 MHz to 300 GHz) are different. They travel in a straight line directly from the transmitter to the receiver. We call this line of sight (LOS). The receiving antenna must be able to "see" the transmitting antenna. Since most of our common wireless devices use LOS propagation, it is useful to know the physics of how this works.

Line of Sight

LOS propagation is a straight line path from TX antenna to RX antenna. If any objects are in the way, the signal will be blocked or at least greatly attenuated. A building may entirely block the transmission. A patch of trees may let some signal through but at a greatly reduced level. In longer distance transmissions, the earth will block the signal due to its curvature. You can actually figure out how far a signal can travel before the earth blocks it (refer to **Figure 1**):

$d = \sqrt{2h}$

Here, *d* is the distance to the horizon, while *h* is the height of the antenna in feet. If the antenna is 30 feet high, the signal will travel:

$d = \sqrt{2(30)} = \sqrt{60} = 7.75$ miles

At that point, the earth will block the signal. If the receiving antenna beyond the horizon is high enough, however, the signal may get through. You can use this next formula to determine the total range D given the height of the transmitting antenna h_t and the height of the receiving antenna h_r. in feet:

$D = \sqrt{2h_t} + \sqrt{2h_r}$

Assume heights of 100 and 70. Total range is:

$D = \sqrt{2h_t} + \sqrt{2h_r} = \sqrt{2(100)} + \sqrt{2(70)} = \sqrt{200} + \sqrt{140} = 14.14 + 11.8 = 26$ miles

Keep in mind that these signals can also be reflected off objects on the way to their destination. Signals can be reflected from cars, planes, buildings, water towers, and any other metallic object. This leads to multiple versions of the original signal reaching the receiving antenna at different times. This is called multipath propagation. The result at the receiver is that these signals can add or subtract to/from one another, creating an effect called fading.

Path Loss

As a radio signal travels through space, it gets weaker. This is called free space path loss (FSPL). It can be estimated by using what is called the Friis formula:

$P_r = P_t G_t G_r \lambda^2 / 16\pi^2 d^2$

This formula calculates the power that reaches the receiver. The different quantities are transmitted power (P_t), received power (P_r), transmitter antenna gain (G_t), and receiver antenna gain (G_r); d is the distance or range and wavelength (λ). The power is in watts and antenna gains are power ratios. Antenna gains are one assuming an isotropic source (spherical radiation pattern). For a dipole or its equivalent, the power ratio is 1.64. Both antennas

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/june2016-Open-Communication-how-wireless-signals-propagate**.

are assumed to have the same polarization.

Note a key point in the formula. The received power is greater for signals of longer wavelengths (λ). Wavelength is related to frequency according to the formula:

$\lambda = 300/f_{MHz}$

Here, wavelength is in meters and f is in MHz. A 900 MHz signal has a wavelength of:

$\lambda = 300/900 = 0.333$ meter Or 33.3 cm

Therefore, according to the Friis formula, the higher the frequency, the lower the wavelength, and the smaller the received power for a given transmit power and antenna gains. All factors being equal, a 900 MHz signal will travel farther than a 2.4 GHz signal. This is the main take-away of the Friis formula.

A better way to determine FSPL is to use this next formula; the attenuation is in dB:

FSPL (dB) = 32.45 + 20log(f) + 20log(d)

The frequency (f) is in MHz and the distance (d) is in kilometers (km).

As an example, what is the FSPL for a 915 MHz signal at 100 feet? Since there are 3.28 feet per meter, 100 feet is 100/3.28 = 30.5 meters or 0.0305 km.

FSPL(dB) = 32.45 + 20log(915) + 20log(0.0305) = 32.45 + 59.2 – 30.3 = 61.35 dB

To make this formula useful, you need to consider some other factors like transmitter power (P_t). Transmit power is usually expressed in dBm (milliwatt reference). For instance, assume a power of 500 mW:

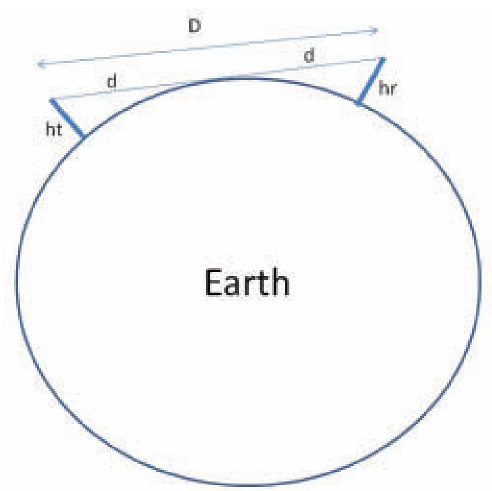


FIGURE 1. Line of sight range is limited by the curvature of the earth and antenna heights.

dBm = 10log (P_t/1mW) = 10log(500) = 27 dBm

Now that you know the path loss and the transmitter power, you can calculate the received power. P_r will also be in dBm:

$P_r = P_t$ – FSPL = 27 – 61.35 = - 34.35 dBm

We need to consider the antenna gains. A dipole or its equivalent has a power ratio gain of 1.64. In decibels, this is a gain of 2.15 dB. Assume that the transmitter and receiver use a dipole. The received power is:

$P_r = P_t + G_t + G_r$ – FSPL = 27 + 2.15 + 2.15 – 61.35 = - 30 dBm

Now, let's consider receiver sensitivity (R). R is a common specification of all wireless receivers. It is a measure of the smallest signal the receiver can hear. The unit of measurement is given in – dBm. Let's assume a value of -94 dBm. Since -30 dBm is a greater power level than -94 dBm, the receiver will get signal sufficient to perform properly with extra margin.

You can also compute the maximum path loss for this combination using the transmit power, receiver sensitivity, and antenna gains:

FSPL (max) = $P_t + G_t + G_r$ – R = 27 + 4.3 – (- 94) = 125.3 dB

This formula is what is called the link budget. You can play around with the various factors to see how things will work.

You can also algebraically rearrange the FSPL formula and calculate the maximum possible range (d) for this example. Here is the mind-numbing calculation:

FSPL (dB) = 32.45 + 20log(f) + 20log(d)

20log(d) = -FSPL (dB) - 32.45 - 20log(f)

d = $\log^{-1}$[FSPL (dB)32.45 - 20log(f)]/20

d = $\log^{-1}$ [125.3 - 32.45 - 20log(915)]/20

d = $\log^{-1}$ [125.3 - 32.45 - 59.2]/20

d = $\log^{-1}$ [1.6825] = 48 km

The link budget assumes a totally clear path and no interference. Actual performance will probably be much less.

Anyway, you can use these formulas to get an approximation of what a proposed wireless system will do. **NV**

■ BY CHUCK HELLEBUYCK

Using 3D printers for practical projects on your workbench.

Adding an LCD and SD Card to a 3D Printer

Just about every 3D printer starts off being controlled by a USB connection. At the heart of most 3D printers is an Arduino based design. The Arduino receives the G-Code serially as it controls the hot end and step height to produce the 3D print. However, this can be a pain to have a computer tied up just for controlling the 3D printer. It would be nice to break away from the computer and run it independently. It turns out that this is actually easy to do on most 3D printers.

Marlin Firmware

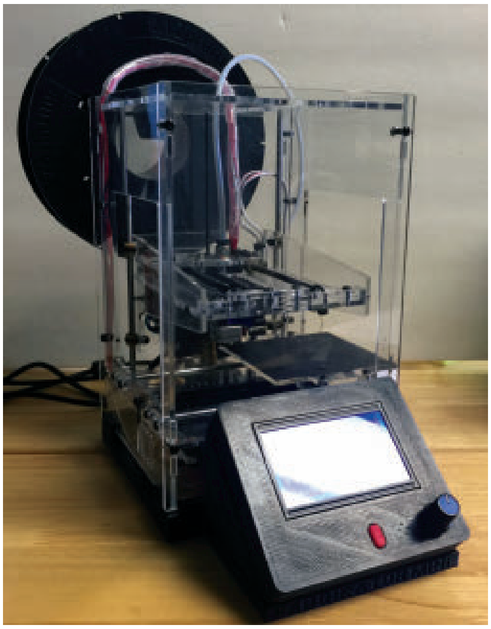
The Fabrikator Mini 3D printer I've shown in previous articles is incredibly handy for electronics hobbyists. It can print fine detail and is small enough to fit on your bench. It's great for making small knobs, brackets, and other support parts. Designs can be printed in pieces as well, and glued or fused together.

One of the drawbacks of this little printer is the need to have it connected to a USB port to print. Fortunately, the printer runs the popular Marlin firmware, which has all the software pre-installed to add an LCD/SD module.

The Marlin firmware just needs one line uncommented in the code to enable the LCD/SD card function for this display. There are several LCD modules listed in the firmware, so you have to pick the right one:

```
// The RepRapDiscount FULL GRAPHIC Smart Controller
(quadratic white PCB)
// http://reprap.org/wiki/RepRapDiscount_Full_Graphic_
Smart_Controller
//
// ==> REMEMBER TO INSTALL U8glib to your ARDUINO
library folder:
// http://code.google.com/p/u8glib/wiki/u8glib
#define REPRAP_DISCOUNT_FULL_GRAPHIC_SMART_CONTROLLER
```

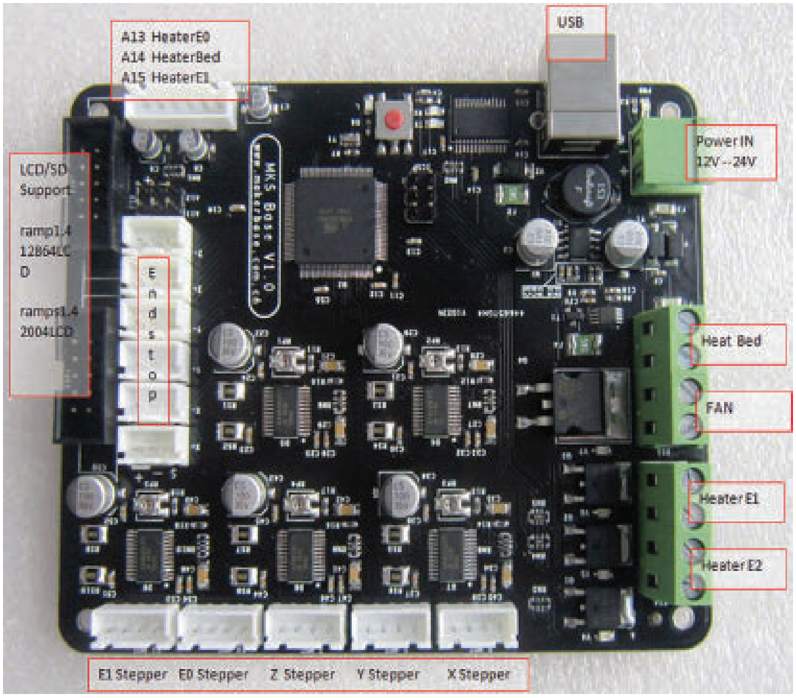
Figure 3 - Configuration.h uncommented line.



■ FIGURE 1 - Fabrikator Mini with LCD/SD card module.

The Configuration.h file in the Marlin.ino sketch is the file that needs to be modified. You just uncomment the line for the RepRap Discount Full Graphic Discount Controller. The comments also point out that you need to install the u8glib library if it's not already installed. You then just upload the new

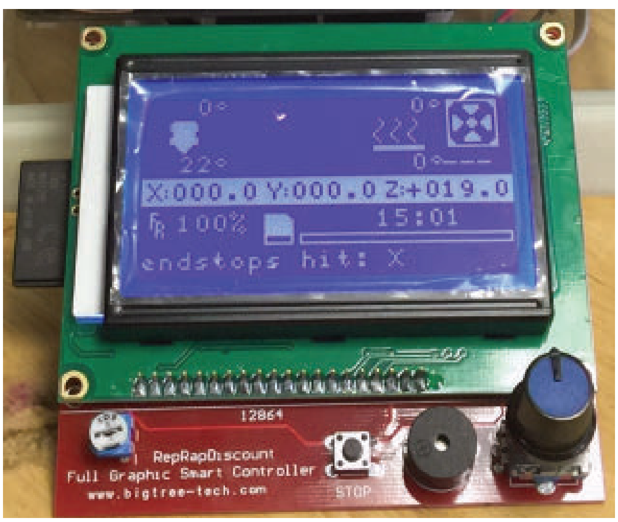
Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/june2016-Practical3DPrinting**.



■ **FIGURE 2. MKS-BASE control board.**

firmware to the Fabrikator Mini and you are ready to use it.

You will need to connect the USB cable from your computer to the Fabrikator Mini to program the firmware with the Arduino IDE (integrated development environment). After that, you can disconnect the cable and hopefully will never have to use it again.



■ **FIGURE 4. RepRap Discount Full Graphic Smart Controller.**

Hardware

The hardware that controls the printer is a variation of the RepRap Arduino Mega Pololu Shield, or RAMPS. It's an MKS-BASE board which is an Arduino and RAMPS combined into a single circuit board. The advantage to this is that just about any RAMPS LCD/SD module can be connected through two ribbon cables to give you the LCD/SD feature.

Installation

The connectors on the MKS-BASE for the LCD are actually backwards. The locator is therefore on the wrong side. This requires you to either reposition the 10-pin ribbon cable connector in the opposite way or (as I did) cut off the locator.

I 3D printed a stand for the LCD and also found a case on **Thingiverse.com** for the display I chose, which was the RepRap controller. This has the SD card on the back and when installed, gives me full control over the Fabrikator Mini and lets me print from the SD card without a computer connected.

Packaging

The case and base are too big to print on the Mini, so I used one of my larger printers to create the designs. You can also cut the designs into sections using many different software tools such as Netfabb Basic or even Tinkercad if you know how to use the "hole" tool and multiple copies of the design. You can also send the designs to a company like Shapeways to print them for a fee and get really great quality prints.

■ **FIGURE 5. LCD display.**

The knob can be printed, but it usually comes with the display module. The knob is a multi-turn pot with a switch, so you turn the knob to find the menu selection, push it to click the switch, and select the option you want. I haven't connected a USB cable to mine since I installed the LCD.

Printing

To use it, you do the normal 3D printing steps of:

1) Download the .stl file or export it from your design software.
2) Import the .stl and slice it with your favorite slicing software; I prefer Simplify3D.
3) Save the G-Code file to the SD card.

Insert the SD card (upside down) into the display module and then use the knob to find the file from the SD card menu and click the knob to start your print. That's all you have to do.

Conclusion

I put together a full YouTube video showing all the steps to do this at **https://youtu.be/IEodObbV7WA**. You can follow along, and I even show how to use it for the first time.

Hopefully, this article and the video will help you get your Fabrikator Mini running without a computer. If you don't have this particular 3D printer, don't worry. Many 3D printers use the same type of firmware setup and hardware connections. **NV**

Resources

Check out my website and blog:
www.elproducts.com

My YouTube channel:
www.youtube.com/elproducts

My 3D designs:
www.thingiverse.com/elproducts/designs

Tinkercad:
www.tinkercad.com

Fabrikator Mini:
Hobbyking.com

 Continued from page 17

flicker) to step #10 slow (ember glow).

For fire simulation, it is best to use fill and flicker light using multiple FECs and light sources. Fill light is typically a complementary light that provides an increased level of ambient light and moderates the depth of the flicker trough. The FEC is capable of a very slow rate of change in all the modes to provide fill light with more of a surging and fading glow appearance than a flicker.

For more dynamic or a faster flicker appearance, multiple FECs in different modes and/or speeds can be used with multiple LED light sources. For burning coal and ember glow simulation, the slower speeds (6-10) of the FEC may be used in any of the modes.

The FEC is for use with LED light sources only. The FEC operates from five to 24 volts DC, and a maximum of 1.5 amps at 5-12 volts and 0.75 amps at 24 volts, 18 watts maximum.

The FEC retains the last setting when turned off or disconnected. The FEC has three tactile buttons:

1) Mode (six total dynamic, one static)

2) Speed - On/Off (10 speed steps and 12 steps of dimming, three second hold for On/Off)

3) Light: Used to change to static dimming mode only (10%-95% dimming range, 12 steps via the Speed - On/Off button)

Applications can include:

- Themed entertainment and dark rides
- Theatrical and decor candles and torches
- Vintage lantern and oil lamp replicas
- Faux fireplaces and glowing coal and embers
- Props and stage scenery fire simulation

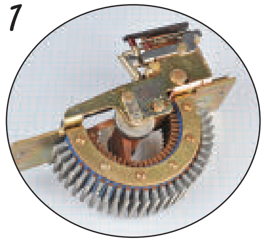
For more information, contact:
J2 LED Lighting, LLC
www.j2ledlighting.com

ELECTRONET

Name that Part!

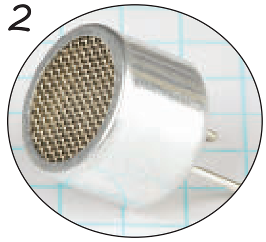
Try this photo quiz and see how many parts you can identify. Some parts date back to the 1950s and earlier, while others can be found at your local RadioShack. For scale, the blue background grid contains 1/4 inch squares. Keep in the mind the photos have been sized to fit the layout. The correct answers can be found on page 66. Good luck!

1



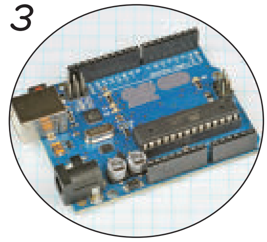
a. Stepping Switch
b. Toggle Switch
c. Wafer Switch

2



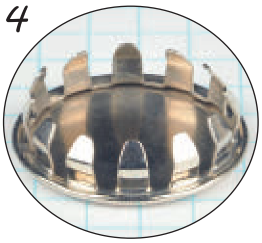
a. RF Filter Screen
b. Micro CPU Fan
c. Ultrasonic Transducer

3



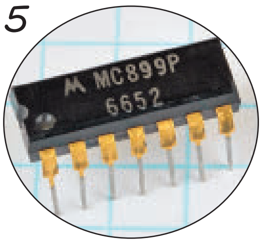
a. Arduino Mega
b. Arduino Micro
c. Arduino Uno

4



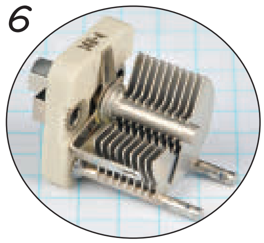
a. Hole Plug
b. Solder Lugs
c. Wire Divider

5



a. TTL IC
b. RTL IC
c. CMOS IC

6

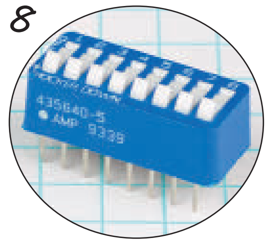


a. Variable Capacitor
b. Variable Inductor
c. Comb Filter

7

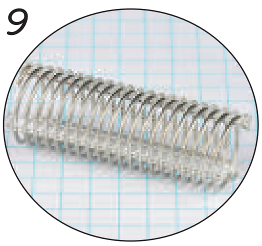
a. Type N
b. TNC
c. BNC

8



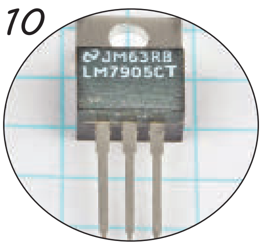
a. DIP Switch
b. Slide Switch
c. Tactile Switch

9



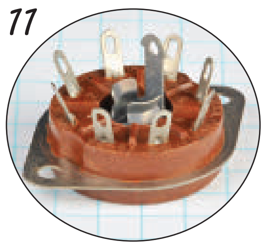
a. Airdux Inductor
b. Spiral Waveguide
c. Coil Spring

10



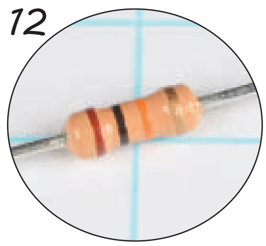
a. +9V Regulator
b. -5V Regulator
c. +5V Regulator

11



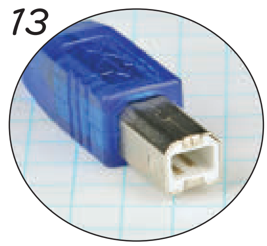
a. Seven-pin Tube Socket
b. Loctal Tube Socket
c. 6SN7 Tube Socket

12



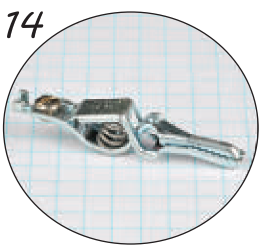
a. 10K ±5% Resistor
b. 10K ±10% Resistor
c. 10K ±1% Resistor

13



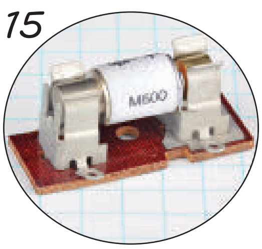
a. USB Type A
b. Mini HDMI
c. USB Type B

14



a. Micro Alligator Clip
b. Salamander Clip
c. Crocodile Clip

15

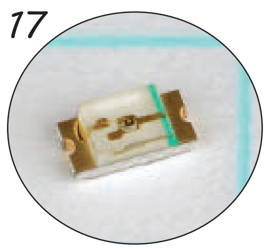


a. M500 Fuse
b. 1N1084 Rectifier
c. M500 Shunt

16

a. Banana Plug
b. Pin Plug
c. RCA Plug

17



a. SMD Inductor
b. SMD Capacitor
c. SMD LED

18

a. SD Memory
b. microSD Memory
c. CompactFlash Card

Presented by David Goodsell

Scoring: 0-7 = Novice 8-15 = Good 16-18 = Expert

 Continued from page 7

cents each, new in the box. I used surplus Eico output transformers, a power transformer, and filter choke that I already had, and I wound a four winding transformer for the tube filaments. It wasn't nearly as pretty as the one in the picture, though. It worked well, but I had to keep replacing gassy 1625s. I replaced it with a solid-state power amplifier from Southwest Technical Products (I still have it) and sold the tube amplifier at a friend's garage sale.

Also in the '60s, I was half owner of a Heathkit analog computer like the one shown in the May issue article on page 40. I modified my Eico oscilloscope to provide a direct coupled input to the X axis so I could demonstrate bouncing balls, etc. In the daytime, I worked with flight simulators that were connected to a whole room full of Pace 231R analog computer consoles.

A short time later, we installed a couple of TR-48 solid-state analog computers, and a whole bunch of interfacing electronics and I/O devices in a modified B-52.

Jerry Nicholson

Thanks Jerry. Nice to hear you shared many of the same experiences.

Bryan Bergeron

As far as a hobby goes, I much prefer tube equipment to late model electronics. The preferred models were H.H. Scott, Zenith, Philco, and others; of course, Collins and Hallicrafters for amateur use were top of the line. Nowadays, newer tube amps are being sold for astronomical prices simply because they are "tube type."

It should be "buyer beware" when purchasing older equipment. For the discerning hobbyist, it's an enjoyable endeavor — much better than newer microprocessor based items in my opinion, and much more durable. No surface-mount parts for me!!

Mike Jobe
Jobe Electronix
Marion, IL

Wow! Quite an adventure, Mike. Yes, when I was a lad, I drooled over a Collins KWM-2. I had a lowly HeathKit SB-102 (after a DX 60B). Yes, those were the days of electronics. No microprocessors. Fix anything with simple tools.

73,
Bryan Bergeron

Restoring Faith

I was reading Developing Perspectives in a recent issue, and I would like to tell you about my endeavors in repairing antique electronics. I am currently restoring an Atwater Kent Model 70 for a guy who bought it at a antique store. The radio was made in 1930 and weighs about 60 lbs. It has a big heavy chassis made of heavy 20 gauge steel and has eight tubes.

The radio is a TRF design with four stages and 45s in a push-pull output. This drives a heavy 25 lb electro-dynamic speaker. The guy dropped the chassis and speaker on the pavement while transporting, as the chassis and speaker are not fastened with bolts on this model.

Luckily, he did little permanent damage other than tearing the speaker wires off of the speaker.

All in all, it had a straightforward design and was well laid out with rugged components. No microprocessors here!! I have been restoring these antiques for about 40 years, so they can be a challenge in finding parts, mystery problems, and such. However, unlike new electronics that fail in a very short time, these beasts have been around since the '20s and have held up well for their age.

The pinnacle of tube radio design would probably be middle 1940s, although there was a Zenith Stratosphere made in 1937 which was one hell of a radio, Not many sold at $750 each, but if you have one today, they are $10,000!! (Almost a news item when one is found these days!)

TECHFORUM
READER - TO - READER

All New TECHFORUM Online At www.nutsvolts.com/tech-forum

>>> QUESTIONS

Pool Bot Timer

My pool bot timer quit. The transformer is good but the timer is not. I would like to build a timer to switch a relay off and back on every 15 seconds and/or 30 seconds.

#6161 **Terry Arnall**
Hayward, CA

Wi-Fi For The Pi

Could someone please tell me if the ESP8266.01 module is compatible with the Raspberry Pi? And/or where can I find a Wi-Fi and GPS module that are?

#6162 **Timothy Harner**
via email

Cats In The Crystal

I inherited an antique "cat's whisker" crystal radio set. Everything seems to be intact except the actual crystal that the cat's whisker touches. Any ideas on where I can find a replacement crystal?

#6163 **Jeffery Payne**
Newport News, VA

PD Race Timer

Does anyone have a schematic for a pinewood derby finish line race timer? I would prefer to build something that doesn't use a microcontroller, if at all possible.

#6164 **Kirk Bateman**
Minneapolis, MN

Dog Bark Detector

I have a very attentive dog who barks immediately when someone is in the backyard or at the front door. This is great for alerting me to visitors or possible intruders, but I really can't hear as well as I used to and my dog has a rather quiet and high-pitched bark that I mostly can't hear — especially if the TV is loud. I would like plans or a design for a "dog bark detector" to blink a light by my chair when the dog is barking so I don't miss these "alerts." Does anyone have a circuit or schematic for such a thing?

#6165 **Irvin Lynch**
Columbia, MO

>>> ANSWERS

[#12151 - December 2015]

Reset Timer

A router and modem at a remote location periodically lock up, requiring a reset by unplugging to restart. Is there a simple circuit I could use to drop a relay out for about 30 seconds, every 24 hours? The relay contacts would be wired to drop out the power circuit to the devices. That way, when it does lock up, at least it would be reset again within a day.

#1 Get an inexpensive 24 hour timer and adjust it so it is on for as much of the 24 hours as you can. If that provides a short enough off time, you are done. If not, you can do the following. Make a 555 monostable timer circuit to energize a relay for about 30 sec. Circuits to trigger a relay using a 555 can be found in many places. Power the 555 circuit using a (surplus) wall cube plugged into your 24 hr timer.

Power your equipment through the relay contacts (but not through the timer) so that when the relay is in the de-energized state, your equipment is on. When the 24 hour timer turns on, the 555 timer will go high for about 30sec, energizing the relay which turns off your equipment for a short time.

B H Suits
Houghton, MI

#2 You can try this particular timer from Amazon: **www.amazon.com/Digital-Programmable-Socket-switch-Energy-Saving/dp/B00WHPNON6/ref=zg_bs_495340_1** Or, Google for household lamp timer; there are many available.

Jerry R.
Long Island, NY

#3 I would consider a simple PIC chip to count the power (50 or 60 Hz) line frequency. It would increment a variable for the seconds, that increments another variable for the minutes and another for the hours. At the 24 hours count, set a pin high to turn on a relay to shut down the modem and router, and clear all the counters. At the 30 second count, set the pin back low for the relay to drop out and power up the gear. The reboot will occur every 24 hours after the chip is powered. A power failure will restart the timer when the power comes back and the cycle will continue. You can change the relay trigger points and intervals to any time period(s) you like! Need more info, just ask!

Len Powell
Finksburg, MD

#4 You could use this timer designed for resetting a router on a

Send all questions and answers by email to **forum@nutsvolts.com**
or via the online form at **www.nutsvolts.com/tech-forum**

timed schedule. It is a plug-in timer! **www.amazon.com/NetReset-NR-1000US-Automated-Cycler-Routers/dp/B00HUEU9H8**

Len Powell
Finksburg, MD

[#4162 - April 2016]
Coils and Magnets and Turns ... Oh My!

I have a question about an article I saw where they had a platter of coils and a platter of magnets. They spun the magnets over the coils producing energy. My question is: How do you determine the correct wire size and number of turns, size and the number of coils, and the size and number of magnets to get to a desired voltage and amperage?

Is there a best way to figure out what combination works?

I appreciate any direction.

#1 You can find the American Wire Gauge (AWG) from this formula that I found in the Magnetics Incorporated Ferrite Core catalog and handbook:

AWG = -4.31*ln(1.889*I/C)

where

I = the AC or DC current

C = the current density in amperes/cm2; 400 is conservative or you can use 800.

To do any calculations, you will need to know the magnetic strength of the magnet, symbolized by H and measured in Oersteds. The magnet manufacturer should be able to give you that info. The H varies with distance so you need that data and how far the coil (I am assuming solenoid coil) is from the magnet. Given the number of turns in the coil and the speed of the magnet, it is possible to calculate the voltage output. That is a complicated procedure and I don't know how to do it.

For a one-off project, it may be better to do trial and error: Choose a core, wind a coil, spin the magnet, and measure the voltage. You will need an oscilloscope. Add or remove turns as needed; pretty straightforward approach! I think the coil should not be much bigger than the magnet; if you have a magnet for each coil and connect the coils in series, the voltages will add. However, you cannot produce a DC voltage that way. You could use a diode to produce pulsating DC to charge a battery.

Russell Kincaid
Milford, NH

#2 This is how I would do it, but I'm devious: Start by figuring out how much voltage and current output you need. Mark these up by 30-50%. Get a brushless DC motor that runs from that much voltage and current. Take out the motor driver circuit and replace it with a polyphase rectifier (sounds intimidating but really just three to six diodes). You are done.

Charles Veres
Hollywood, FL

[#4163 - April 2016]
Power Supply Replacement

I have a set of computer speakers which have become separated from their power supply.

Can you help me work out what power supply I need to replace the missing unit?

It may be helpful (to your other readers as well) if you could suggest a general procedure for this sort of situation that avoids blowing up the equipment.

In my case, I have a head start as the power jack plug is labeled with "DC in" and the polarity (center negative). It does not give the voltage though!

If there were no markings at all I would have to work out AC or DC as well. The connector type should be obvious of course, but I could imagine that even this could be difficult on occasion.

There are three possible solutions.

First and easiest is to contact the speaker brand or supplier with the model number. They will give you the exact supply information and may even offer to sell you a replacement.

Second, open the speaker that has the DC input and look at the electrolytic capacitors. Search their labels for their WVDC rating, and multiply by 0.8 as a safety factor. For example, if the rating was 15 WVDC, then the maximum power supply voltage would be 12 VDC which is very common. The current rating depends on the load when the volume is maximum, but a good guess to begin with is 0.5A (500 mA). Get the new power supply, test the speakers with a signal, and increase the volume. If the sound gets muddy or distorted, then use a higher current rating, probably adding another 0.1A or more. The speaker will not use any excess current, but the new power supply must not drop the voltage at the highest volume.

Third and most diffcult is to use a variable DC power supply. Start with 6 VDC and power up the speakers. If the volume is too low and a higher setting distorts the sound, the circuit is limited (clipping) and needs more voltage. Apply increments of 0.5 VDC and repeat the test until the sound is good at all levels, then note the applied voltage. If it is a strange voltage (for example, 8.5V) then add another 0.5V and test again.

If satisfactory, then buy or build a new power supply with that voltage and add 0.1A (100 mA) over the current measured on the variable power supply. Good luck and have fun!

Raymond J Ramirez
Bayamon, PR

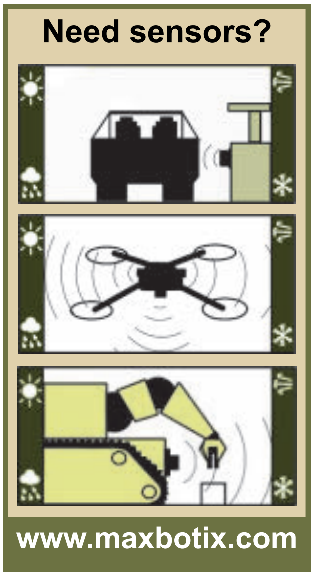
CLASSIFIEDS

LIGHTING

EDUCATION

HARDWARE WANTED

ROBOTICS



ANSWERS TO NAME THAT PART on page 62:

1a, 2c, 3c, 4a, 5b, 6a, 7c, 8a, 9a, 10b, 11b, 12a, 13c, 14c, 15b, 16a, 17c, 18b

AUDIO/VIDEO

ADvertiser INDEX

LOOK FOR SEARCH FOR FIND **Find your favorite advertisers here!**


"Fetch newspaper, Rex!"
THINK
BUILD
CONNECT